RetroMagazine
Anno 2 - Numero 5

MARZO 2018 - WWW.RETROMAGAZINE.NET

IN EVIDENZA IN QUESTO NUMERO

RetroMagazine cresce!

di Francesco Fiorentini

Si respira aria di novita' nella redazione di RetroMagazine. Grazie anche al successo dei numeri sin ora pubblicati, abbiamo deciso di allargare i nostri orizzonti e di creare un sito web ed una pagina Facebook interamente dedicati alla rivista.

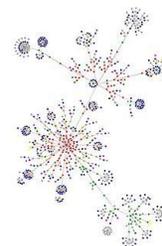
Nella pagina Facebook, che si chiama ovviamente **RetroMagazine**, posteremo le anticipazioni degli articoli che usciranno nei prossimi numeri ed attenderemo i vostri commenti. Vi invitiamo fin da ora a visitarla, se non lo avete ancora fatto, ed a mettervi un like cosi' da ricevere gli aggiornamenti che di volta in volta posteremo. La pagina sara' la cartina di tornasole dell'apprezzamento dei lettori. Sentitevi quindi liberi di commentare e/o di avanzare richieste!

Inoltre, come probabilmente i lettori piu' attenti avranno gia' avuto modo di notare sul numero 4, il sito web, raggiungibile alll'indirizzo www.retromagazine.net, diventera' l'archivio ufficiale della rivista. Li' troverete i riferimenti ai numeri gia' pubblicati insieme ad altro materiale che di volta in volta condivideremo.

Ovviamente, come dice un famoso adagio, Roma non fu fatta in un giorno, entrambe le risorse sopra elencate sono ancora in fase di costruzione; non stupitevi quindi se per il momento i contenuti saranno scarsi, l'obiettivo e' quello di arricchirli entrambi e farli crescere proprio come abbiamo fatto con la rivista!

Voglio pero' rassicurare i nostri lettori. RetroMagazine e' nato come una rivista e come tale continuera' la sua pubblicazione. L'impegno di tutta la redazione e' rivolto principalmente a scrivere articoli nuovi da pubblicare sulla rivista e ci sentiamo di assicurarvi che le idee e gli argomenti certamente non mancano! E le sorprese nemmeno, continuate a seguirci per esserne partecipi.

Il numero che vi apprestate a leggere vede il ritorno di RetroMath, un articolo su come incorporare i file SID in Assembly a firma di Marco Pistorio, la seconda parte della programmazione della console Atari 2600 e la presentazione di un progetto tutto italiano a cura di **Fabrizio Caruso**!

Vi anticipo sin da ora che e' nostra intenzione dare sempre piu' spazio ai progetti legati al retrocomputing, quindi se volete far conoscere il vostro, contattateci; il modo lo conoscete. A questo punto non mi resta altro che augurarvi buona lettura!

RetroMath: Grafi, cammini e giochi

Torna la rubrica RetroMath: cos'è un grafo, a cosa serve, è possibile utilizzarlo per schematizzare e risolvere alcuni problemi di vita quotidiana? A Giuseppe l'onere e l'onore della risposta.

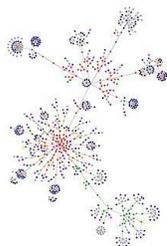
Articolo a pagina 2

Leader Board - (C64)

E' il 1986 e Access Software se ne esce con un gioco del Golf per Commodore 64 destinato a riscrivere la storia di questo sport su tutti gli home computer. Scopriamo insieme a Francesco se il gioco e' ancora attuale o se ha sofferto il passare degli anni.

Articolo a pagina 15

RetroMath:
Grafi, cammini e giochi

di Giuseppe Fedele

Cos'è un grafo, a cosa serve, è possibile utilizzarlo per schematizzare e risolvere alcuni problemi di vita quotidiana?

Partendo da giochi e rompicapo spieghiamo come passeggiare sui grafi e cosa significa cammino Euleriano e cammino Hamiltoniano e come è complicato e "oneroso" (a volte) camminare sui grafi!!!

Attenzione...è un problema da un milione di dollari!!!

Si racconta che nel 1736 il grande matematico Leonhard Euler (Basilea, 15 aprile 1707 – San Pietroburgo, 18 settembre 1783) si fermò, durante uno dei suoi viaggi, a Königsberg (oggi nota col nome di Kaliningrad, Figura 1).

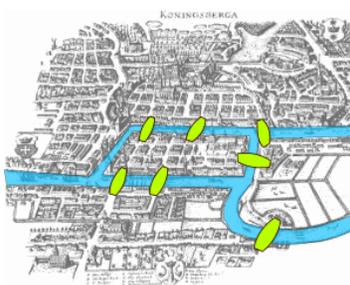



Figura 1. L'antica città di Königsberg.

Gli abitanti di Königsberg, la domenica, erano soliti passeggiare per la loro città cercando di risolvere il seguente problema (descritto dallo stesso Eulero):

"Il problema, che per quanto so, è molto conosciuto, si pone come segue: nella città di Königsberg, in Prussia, c'è un'isola, chiamata Kneiphof, circondata dai rami del fiume Pregel. Ci sono sette ponti che attraversano i due rami del fiume. La questione consiste nel determinare se una persona può fare una passeggiata in modo da attraversare ciascuno dei ponti una volta sola. Sono stato informato che mentre alcuni negavano la possibilità di farlo e altri lo dubitavano, nessuno sosteneva che fosse realmente possibile".

Una soluzione naïve è quella di elencare tutte le passeggiate possibili, trovando il cammino che soddisfa il problema oppure constatando che tale passeggiata non esiste. Chiaramente l'approccio "bruto" (ovvero una soluzione tramite ricerca esaustiva di tutte le possibili soluzioni) non poteva essere quella di un grande matematico come Eulero, e questo per almeno due motivi: i percorsi possibili sono tantissimi e così facendo si risolverebbe il problema specifico perdendo di vista quello più generale.

Eulero dimostrò che non era possibile e basò la sua risposta sul seguente ragionamento: tracciamo uno schema della città mediante quattro punti A, B, C, D (che corrispondono alle quattro parti della città) e indichiamo con a, b, c, d, e, f, g i sette ponti (Figura 2). Il problema da risolvere è equivalente a capire se, partendo da uno dei quattro punti, è possibile tracciare un itinerario che contenga tutte le curve una volta sola. Se questo fosse possibile il numero di linee per ciascun punto dovrebbe essere pari e invece tutti i punti hanno un numero dispari di linee. Quindi il problema non ha una soluzione.

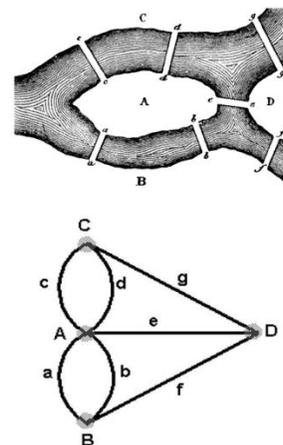



Figura 2. Grafo associato al problema dei 7 ponti.

I ponti furono distrutti durante la Seconda Guerra Mondiale, ma l'impostazione al problema ideata da Eulero consentì la nascita di quella che conosciamo oggi come Teoria dei Grafi.

Nozioni di base sulla Teoria dei Grafi

Un grafo G è una coppia $G = (V, E)$, dove V è un insieme finito e non vuoto, detto insieme dei vertici (o nodi), ed E è l'insieme dei lati (o archi). In generale, gli archi rappresentano una relazione tra una coppia di nodi; se tale coppia è ordinata, cioè se gli archi hanno una testa (o nodi di arrivo) ed una coda (o nodo di partenza), il grafo si dice **orientato**. Nel caso in cui le coppie di nodi presenti nel grafo non siano ordinate il grafo di dice **non orientato**.

Due archi si dicono **adiacenti** se esiste un nodo in comune ad entrambi (nel grafo non orientato gli archi (3,4) e (4,5) sono adiacenti, non lo sono viceversa nel grafo orientato).

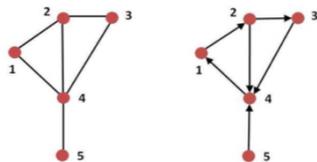



Figura 3. Grafo non orientato (a sinistra) e orientato (a destra).

In un grafo non orientato è detto **cammino** un insieme di archi $(a_1, a_2, \dots, a_n)$ tale per cui a_i e a_{i+1} sono adiacenti. In un grafo orientato un cammino richiede che la sequenza di archi sia tale che la testa di un arco coincida con la coda del successivo.

Un nodo si dice **adiacente** ad un altro nodo se esiste un arco che li congiunge.

Un grafo tale per cui per ogni coppia di nodi esiste un cammino che li unisce è detto **grafo connesso**.

Altre due definizioni importanti nell'ambito della teoria dei grafi sono i cammini euleriani e i cammini hamiltoniani. Un cammino che passa esattamente una volta per ogni arco è detto **cammino euleriano**; per cui un grafo che ammette un cammino euleriano è detto **grafo euleriano**. Dualmente un cammino che passa esattamente una volta sola per ogni nodo è detto **cammino hamiltoniano** e il corrispondente grafo è detto **grafo hamiltoniano**.

Esistono diversi modi per memorizzare un grafo su un calcolatore. La **matrice di connessione** utilizza una matrice quadrata di dimensione $n \times n$, se n è il numero di nodi del grafo, in cui l'elemento (i, j) vale 1 se esiste un arco che connette il nodo i con il nodo j e 0 altrimenti. Nel caso di grafi non orientati la matrice sarà simmetrica, ovvero l'elemento (i, j) è uguale all'elemento (j, i).

La **matrice di incidenza** è invece una matrice di dimensione $n \times m$, dove m è il numero degli archi del grafo. In questo caso l'elemento (i, j) della matrice vale 1 se l'arco j incide sul nodo, 0 altrimenti. Nel caso di grafi orientati il generico elemento (i, j) sarà 1 se l'arco j esce dal nodo i, -1 se l'arco j entra nel nodo i, 0 altrimenti. Infine la **lista di adiacenza** è basata sulla memorizzazione di una lista che contiene, per ogni nodo del grafo, l'elenco dei relativi nodi adiacenti. Nel caso di grafo orientato in Figura 3 le tre strutture sono rappresentate di seguito.

Matrice di connessione:

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	1	0
3	0	0	0	1	0
4	1	0	0	0	0
5	0	0	0	1	0

Matrice di incidenza:

	(1,2)	(2,3)	(2,4)	(3,4)	(4,1)	(5,4)
1	1	0	0	0	-1	0
2	-1	1	1	0	0	0
3	0	-1	0	1	0	0
4	0	0	-1	-1	1	-1
5	0	0	0	0	0	1

Lista di adiacenza:

	Nodi adiacenti
1	2
2	3, 4
3	4
4	1
5	4

Il problema dello scambio dei cavalli

Il problema è stato proposto dal matematico britannico Henry Ernest Dudeney (1857 – 1930) che ha contribuito con i suoi giochi logici alla diffusione della matematica ricreativa.

Data una scacchiera 3×3 e numerate le caselle come in Figura 4, disponiamo i cavalli bianchi nelle caselle 1 e 3 e quelli neri nelle caselle 7 e 9. Il gioco consiste nel cambiare di posto i cavalli (i bianchi in 7 e 9 ed i neri in 1 e 3) spostando un cavallo alla volta secondo le modalità degli scacchi senza avere due cavalli nella stessa casella.

Per risolvere il problema possiamo utilizzare il grafo in Figura 4, dove ad ogni casella associamo un nodo ed esiste un arco tra due nodi se è possibile spostarsi da un nodo all'altro effettuando la mossa del cavallo.

E' evidente che la casella 5 non risulta raggiungibile da nessun cavallo e che una soluzione si ottiene facendo circolare i cavalli secondo lo schema del grafo in modo da non sovrapporli in nessuna delle configurazioni. Partendo come prima mossa dal cavallo bianco nella casella 1, la sequenza delle mosse è:

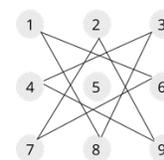
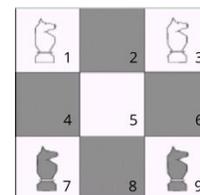



Figura 4. Il problema dei 4 cavalli.

CAVALLO (CB-CN)	DA	A
CB	1	6
CN	7	2
CN	9	4
CB	3	8
CB	8	1
CB	6	7
CN	2	9
CN	4	3
CN	3	8
CB	1	6
CB	7	2
CN	9	4
CN	4	3
CN	8	1
CB	6	7
CB	2	9

Le torri di Hanoi

Si tratta di un gioco matematico inventato dal matematico francese François Édouard Anatole Lucas (1842 – 1891). La torre è formata da una pila di dischi sovrapposti uno sull'altro sul piolo 1 (Figura 5).

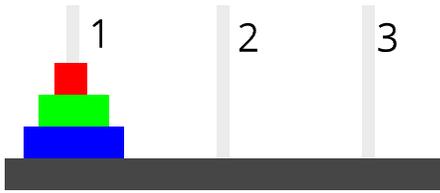



Figura 5. Le torri di Hanoi.

Lo scopo del gioco è spostare i dischi sul piolo 3 in modo da ottenere la stessa pila, aiutandosi con il piolo centrale. Bisogna però seguire la regola secondo cui è possibile spostare soltanto un disco alla volta ed è proibito collocare uno qualsiasi dei dischi su uno più piccolo.

E' possibile dimostrare che se n è il numero di dischi, esiste una sequenza di $2^n - 1$ mosse che permette di spostare l'intera torre in uno dei pioli liberi.

Vediamo adesso come è possibile risolvere il problema, nel caso di $n = 3$, utilizzando la teoria dei grafi descritta fino ad ora. Per prima cosa dobbiamo codificare tutte le possibili mosse in un grafo. Come? Codifichiamo ogni posizione del gioco con una terna (Figura 6):

- Il primo numero indica il piolo in cui si trova il disco più piccolo;
- Il secondo numero indica il piolo in cui si trova il disco medio;
- Il terzo numero indica il piolo in cui si trova il disco più grande.

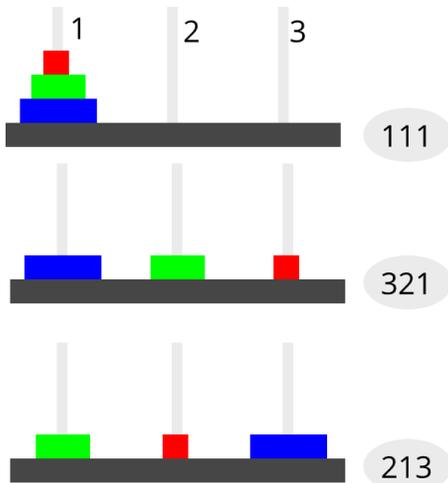



Figura 6. Codifica delle posizioni.

Con un po' di pazienza è quindi possibile codificare tutte le possibili mosse del gioco e creare il grafo in Figura 7.

Il problema, come i più attenti avranno capito, consiste nel trovare un cammino hamiltoniano che parte dal nodo in alto e termina nel nodo in basso a destra. In Figura 8 la curva rossa mostra il cammino cercato.

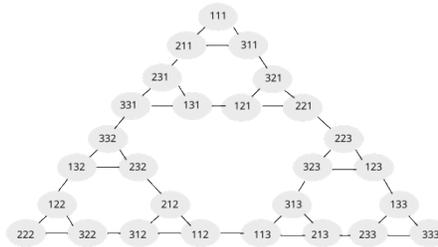



Figura 7. Grafo associato al problema delle Torri di Hanoi.

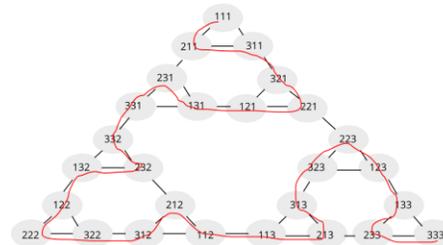



Figura 8. Cammino Hamiltoniano per le Torri di Hanoi.

Una soluzione al problema delle Torri di Hanoi può essere formulata in veste ricorsiva (come fare nel caso di linguaggi che non ammettono la ricorsione???).

Per spostare n dischi dal piolo di sinistra a quello di destra si può procedere come segue:

- Se $n = 1$, spostare il dischetto dal piolo sorgente al piolo destinazione;
- Se $n > 1$ allora:
  - Spostare $n - 1$ dischi dal piolo sorgente al piolo ausiliario;
  - Spostare un disco dal piolo sorgente al piolo destinazione;
  - Spostare $n - 1$ dischi dal piolo ausiliario a quello destinazione.

Il gioco del quadrato $n \times n$

Il gioco consiste nel riempire una griglia quadrata di dimensione $n \times n$ con i numeri che vanno da 1 a n^2, rispettando le seguenti regole:

- Il numero 1 può essere posizionato in una qualsiasi delle n^2 caselle;
- ci si può spostare in orizzontale e verticale saltando due caselle;
- ci si può spostare in diagonale saltando una casella.

Ad esempio, dalla casella contrassegnata con la lettera O (Figura 8) ci si può spostare in orizzontale nella casella L e in diagonale nelle caselle C e W.

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	Q	R	S	T
U	V	W	X	Y

Figura 8. Gioco del quadrato con $n = 5$.

Anche in questo esempio è possibile associare al gioco un grafo in cui i nodi rappresentano le caselle e i cui lati uniscono due nodi se le corrispondenti caselle sono collegabili con una mossa valida.

Il grafo, nel caso di tabella 5×5 è riportato in Figura 9. La soluzione consiste quindi nell'individuare un cammino hamiltoniano nel grafo (a partire dalla posizione in cui si è deciso di inserire il numero 1).

Una soluzione con $n = 5$ e $n = 10$ è mostrata qui di seguito:

1	22	5	2	23
14	11	8	15	12
6	3	18	21	4
9	25	13	10	24
17	20	7	16	19

1	69	66	2	70	65	23	54	64	22
43	18	15	44	19	14	45	20	13	46
67	3	41	68	95	57	71	98	24	53
16	83	76	17	84	77	26	55	63	21
42	35	94	79	72	99	90	58	12	47
75	4	40	86	96	56	85	97	25	52
31	82	73	34	89	78	27	50	62	8
39	36	93	80	37	100	91	59	11	48
74	5	30	87	6	29	88	7	28	51
32	81	38	33	92	60	10	49	61	9

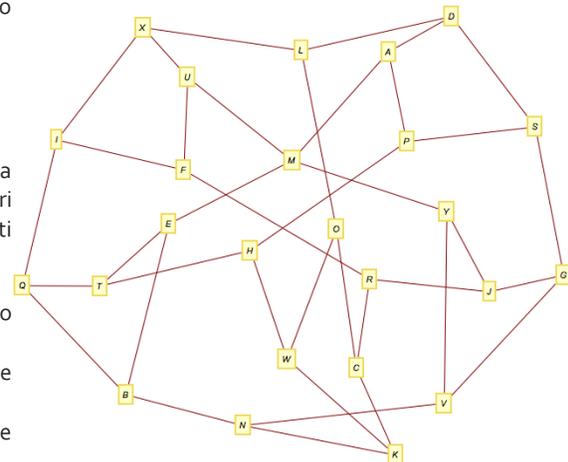



Figura 9. Grafo per il gioco della tabella 5×5.

__Scheda di approfondimento__

Cenni di informatica teorica

Un **problema** è una relazione matematica tra dati di ingresso e dati di uscita.

Una **istanza** di un problema è formata dai dati di un generico ingresso del problema. Una istanza ha associato uno **spazio di ricerca** nel quale cercare una soluzione per tale istanza del problema.

Esempio:

Data una sequenza S di numeri trovare una permutazione che sia ordinata in senso crescente. L'istanza è una sequenza I

$$I = \{4,2,3\}.$$

Lo spazio di ricerca associato ad I è l'insieme di tutte le permutazioni:

$\{2,3,4\}, \{2,4,3\}, \{3,2,4\}, \{3,4,2\}, \{4,2,3\}, \{4,3,2\}$
Una soluzione per l'istanza I è una permutazione di I che sia ordinata in senso crescente.

Risolvere un problema significa trovare un algoritmo che prende in ingresso una istanza I del problema e restituisce una soluzione per I. **Verificare un problema** significa identificare un algoritmo che prende in ingresso una istanza I del problema e un elemento α dello spazio di ricerca associato a I, e verifica se α risolve I.

Classe P. Un problema appartiene alla classe P se esiste un algoritmo di soluzione la cui complessità è polinomiale nelle dimensioni dell'input. Questo significa che un problema in P può essere risolto in tempo polinomiale.

Classe NP. Un problema appartiene alla classe NP se può essere verificato in tempo polinomiale (attenzione verificato non risolto!!!).

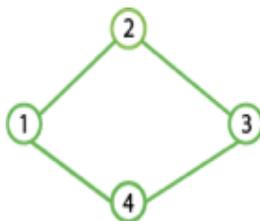
I problemi **NP-Completi** sono i problemi più difficili della classe NP. Se si scoprisse un algoritmo polinomiale per risolverne uno, allora si dimostra che tutti i problemi in NP potrebbero essere risolti in tempo polinomiale. D'altra parte se si scoprisse che per risolvere anche uno solo dei problemi in NP è indispensabile un tempo esponenziale, allora tutti i problemi NP-Completi richiederebbero tale tempo.

Se volete cimentarvi c'è un premio di un milione di dollari per chi riesce a risolvere questo problema **P CONTRO NP**.

(https://it.wikipedia.org/wiki/Problemi_per_il_millennio#P_contro_NP).

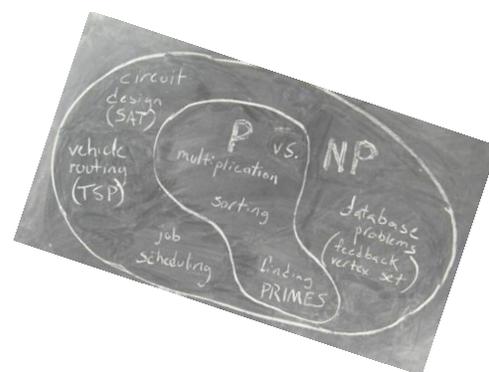
Il premio Clay chiede essenzialmente una dimostrazione che i problemi di tipo NP siano effettivamente distinti da quelli di tipo P ovvero che non esistano scorciatoie polinomiali per risolverli. Se questo fosse trovato per uno specifico problema NP, allora tutti gli NP sarebbero P. In realtà i matematici si aspettano che tutti i problemi di tipo NP siano non-P (anche se nessuno è ancora in grado di provarlo).

Un percorso Hamiltoniano in grafi orientati e non-orientati, è un semplice percorso aperto che contiene tutti i nodi di un dato grafo esattamente una volta. Il problema si riduce a determinare se esiste un percorso aperto che attraversa tutti i nodi del grafo esattamente una volta. La ricerca di percorsi hamiltoniani in un grafo può essere risolta verificando tutte le permutazioni degli n nodi del grafo verificando l'esistenza di un arco tra il nodo corrente e il successivo nel grafo di partenza. In questo modo, basterà verificare per tutte le permutazioni dei nodi se qualcuna di loro rappresenta un percorso Hamiltoniano oppure no.

Il grafo:

è composto da $n = 4$ nodi, le permutazioni di n (che si esprimono come $n!$) sono 4! = 4*3*2*1 = 24. Quindi in un grafo con 4 nodi esistono 24 possibili percorsi. Di questi 24 percorsi solo 4 sono hamiltoniani e sono :

{1,2,3,4},{2,3,4,1},{3,4,1,2},{4,1,2,3}

Sembrerebbe che per risolvere il problema della ricerca di percorsi Hamiltoniani, basti calcolare le permutazioni dei nodi e verificare l'esistenza dei relativi archi nel grafo. Il problema è che le permutazioni da calcolare crescono in modo eccessivo al crescere del numero di nodi. Per esempio, in un grafo con $n = 8$ nodi, le permutazioni possibili sono 40.320, per $n = 10$ le permutazioni sono 3.628.800 mentre per $n = 15$ le permutazioni sono 1.307.674.368.000. Dunque si capisce che il problema della ricerca di percorsi Hamiltoniani in grafi orientati e non-orientati appartiene alla classe di problemi NP-Completi. Sebbene la ricerca di percorsi hamiltoniani è decidibile e quindi computazionalmente risolvibile in teoria, non è risolvibile efficientemente in pratica perché la soluzione richiede quantità eccessive di tempo e memoria anche per gli odierni calcolatori. Attualmente la ricerca di percorsi hamiltoniani viene effettuata mediante algoritmi approssimati conosciuti con il nome di Algoritmi Euristici. Gli algoritmi euristici hanno come caratteristica una alta probabilità di produrre una "buona soluzione" in tempi brevi.

Gli attuali algoritmi euristici possono trovare soluzioni in tempi accettabili anche per problemi concreti ad esempio grafi con milioni di nodi come reti sociali, reti biologiche e reti proteiche, e le soluzioni trovate si discostano del 2-3% rispetto alla soluzione esatta.

Bibliografia:

Dino Mandrioli, Paola Spoletini.
Informatica teorica.
Città Studi.

Emanuele Delucchi, Giovanni Gaiffi, Ludovico Pernazza.
Giochi e percorsi matematici.
Springer.

Peter M. Higgings.
La matematica dei social networks. Una introduzione alla teoria dei grafi.
Dedalo.

Un ringraziamento particolare all'amico e collega **Ing. Giuseppe Agapito** dell'Università Magna Graecia di Catanzaro per gli approfondimenti teorici.

Uso dei files .sid all'interno di programmi assembly per C64

di Marco Pistorio

Parte I – Cosa sono i files .sid

I files .sid sono files che contengono uno o più brani musicali, codificati secondo regole ben precise. Esistono dei repository che contengono files .sid dove è di norma possibile effettuare ricerche e dove è possibile anche effettuare il download del file .sid desiderato, in maniera agevole ed intuitiva. Uno di questi repository si trova all'indirizzo https://www.hvsc.c64.org/

L'High Voltage SID Collection (HVSC) è un progetto hobbistico che organizza i files .sid in un archivio per musicisti e fan. Il lavoro sulla collezione viene svolto nel tempo libero da parte dei membri del team e dei contributori ed è una delle più grandi ed accurate raccolte di musica per computers conosciute.

I vantaggi nell'utilizzare un file .sid all'interno di un programma assembly sono molteplici. Grazie alla loro rigorosa codifica, diventa semplice ed immediato impiegare un certo brano piuttosto che un altro, rendendo di fatto il proprio lavoro di codifica separato da ciò che serve per gestire il brano musicale.

All'interno di ciascun file .sid possiamo individuare tre componenti importanti, che sono: l'header, la routine di inizializzazione e quella di esecuzione del brano. L'header contiene tutta una serie di informazioni utili, tra le quali gli indirizzi di partenza delle suddette routines di inizializzazione e di esecuzione del brano, il numero di brani presenti nel file .sid (solitamente 1 ma non necessariamente), il nome dell'autore, informazioni relative al copyright del brano e molto altro ancora.Per una trattazione più precisa, esaustiva e dettagliata, rimando i lettori interessati a questo link:

https://www.hvsc.c64.org/download/C64Music/DOCUMENTS/SID_file_format.txt

Diversi nostri lettori, ci chiedevano come riuscire ad incorporare un file .sid all'interno di programmi assembly con CBM PRG STUDIO.

E' possibile certamente fare ciò. L'unico inconveniente, utilizzando l'ambiente CBM PRG STUDIO, è che non vengono gestite le informazioni contenute nell'header dei files .sid nativamente. Di conseguenza, nel codice di esempio rimuovo tali informazioni perché inutili, alla luce del loro impiego con CBM PRG STUDIO.

KickAssembler invece, a titolo di esempio, tratta queste informazioni e ne ricava ciò che serve per fruire del file .sid in maniera automatica.

Queste sono le informazioni che si riescono ad ottenere in maniera automatica tramite Kick Assembler esaminando l'header di un file .sid preso ad esempio: "Battlestar_Galactica.sid"

```
SID Data
--------
location=$2000
init=$2000
play=$2003
songs=1
startSong=1
size=$73d
name=Battlestar Galactica
author=Jeroen & Michiel Soede
copyright=1988 SoedeSoft
Additional tech data
--------------------
header=PSID
header version=2
flags=10100
speed=0
startpage=0
pagelength=0
```

In particolare, init e play rappresentano le locazioni iniziali delle routine di inizializzazione e di esecuzione del brano. **Per un maggiore dettaglio delle informazioni presentate rimando al link già citato in precedenza**.

A questo punto ci si potrebbe chiedere se sia meglio scegliere un certo assemblatore piuttosto che un altro. Ma questo articolo non copre questa tematica. Possiamo però senza dubbio affermare che ciascuna scelta presenta pregi e difetti. Ad esempio, CBM PRG STUDIO è closed-source ed è legato alla piattaforma WIN. Kick Assembler è anch'esso closed-source ma è multipiattaforma. ACME invece è open-source, ed ha una sintassi simile a quella di KickAssembler. Mi fermo qui, ma sappiate che ne esistono diversi altri :)

Tornando a CBM PRG STUDIO, e contestualmente a tutti gli altri casi di assemblatori che non riescono a gestire nativamente le informazioni contenuti nell'header del file .sid, come fare per ottenere le preziose informazioni relative in particolare alla routine di inizializzazione (init) e quella di esecuzione del brano (play)?

Ci viene in aiuto a questo punto il sito di HVSC. Effettuando una ricerca del brano e selezionando la voce "Details", otteniamo una schermata come questa, contenente le info che ci interessano (**vedi figura 1**).

Parte II – La "temporizzazione"

Questa è probabilmente la parte più delicata di questa "chiacchierata", insieme alla successiva. E' necessario lanciare la routine di esecuzione del brano a tempo, altrimenti il brano risulterà eseguito troppo velocemente

oppure troppo lentamente. Come effettuare una "temporizzazione" corretta? Dobbiamo usare una sorta di "metronomo".

A questo punto, Vi faccio notare una informazione particolare presente nella figura contenente i dati presentati da HVSC relativamente al brano scelto. Notate la voce "Clock PAL". Cosa vuole dire? Significa che il brano è adatto per essere sincronizzato con la frequenza di refresh del formato video PAL.

Lo schermo video del C64 viene disegnato un certo numero di volte al secondo. E' possibile, mediante una particolare tecnica ("interruzioni di RASTER"), sfruttare l'istante in cui viene disegnata una ben precisa linea dello schermo per lanciare l'esecuzione della routine play del file .sid. Ogni qualvolta verrà ridisegnata tale linea dello schermo, verrà rieseguita anche la routine play del file .sid. Esistono brani impostati per la frequenza **PAL** (frequenza 50 Hz, 625 linee di schermo, sistema Europeo), altri per la frequenza **NTSC** (frequenza 60 Hz, 525 linee di schermo, sistema Americano) etc.

Codice Assembly con sintassi CBM PRG STUDIO. C64 modalità PAL:

```
*=$0801
byte $0b,$08,$0a,$00,$9e,$32,$30,$36,$34,$00,$00,$00
*=$0810
lda #$00      ; inizializzazione sound
tax           ;
tay           ;
jsr $2000     ; address of inizialization subroutine
sei           ; disabilita IRQ
lda #%0111111 ;
sta $dc0d     ; Interrupt control and status register
              ; Bit #0: 1 = Enable interrupts generated by timer A underflow.
              ; Bit #1: 1 = Enable interrupts generated by timer B underflow.
              ; Bit #2: 1 = Enable TOD alarm interrupt.
              ; Bit #3: 1 = Enable interrupts generated by a byte having been
              ;             received/sent via serial shift register.
              ; Bit #4: 1 = Enable interrupts generated by positive edge on FLAG
              ;             pin.
sta $dd0d     ; Interrupt control and status register
              ; Bit #0: 1 = Enable non-maskable interrupts generated by timer A
              ;             underflow.
              ; Bit #1: 1 = Enable non-maskable interrupts generated by timer B
              ;             underflow.
              ; Bit #2: 1 = Enable TOD alarm non-maskable interrupt.
              ; Bit #3: 1 = Enable non-maskable interrupts generated by a byte
              ;             having been received/sent via serial shift register.
              ; Bit #4: 1 = Enable non-maskable interrupts generated by positive
              ;             edge on FLAG pin.
lda #$01      ;
sta $d01a     ; Interrupt control register
              ; Bit #0: 1 = Raster interrupt enabled.
lda #$1b
ldx #$c8      ; Default value for $d016 (Screen control register #2)
ldy #%00010100
sta $d011     ; Raster line to generate interrupt at (bit #8)
stx $d016     ; Screen control register #2
sty $d018     ; Memory setup register
              ; Bits #1-#3 pointer to character memory %010,
              ;            2: $1000-$17FF, 4096-6143.
              ; Bits #4-#7 Pointer to screen memory %0001,
              ;            1: $0400-$07FF, 1024-2047.
lda #<irq
ldx #>irq
ldy #$7e
sta $0314     ; modifica il vettore per l'esecuzione
stx $0315     ; della routine di servizio IRQ alla locazione -irqsty
$d012         ; Raster line to generate interrupt at (bit #0-#7)
lda $dc0d     ; reset per eventuali interrupts rilevati
lda $dd0d     ; reset per eventuali interrupts rilevati
asl $d019     ; Acknowledge raster interrupt
cli           ; abilita IRQ
rts           ; uscita programma
irq
asl $d019     ; Acknowledge raster interrupt
;inc $d020
jsr $2003     ; address of play-sound subroutine
;dec $d020
jmp $ea31     ; salta alla normale routine di servizio IRQ
*=$2000       ; load-address

INCBIN "Battlestar_Galactica.sid",126 ; rimuove headers non necessari
```

Perchè è stato scelto questo sistema di "temporizzazione" tramite raster a discapito di altri sistemi? La risposta è semplice. E' un sistema affidabile, estremamente preciso e già disponibile, a differenza di altri metodi che si potrebbero implementare e gestire, sfruttando oltretutto ulteriori cicli di CPU.

L'esempio fornito insieme a questo articolo nasce per sistemi video PAL ovvero il sistema video in uso in Europa.

E' possibile adattare un brano temporizzato per la frequenza PAL per essere eseguito anche in modalità video NTSC con qualche stratagemma. Uno di questi metodi mi è stato fornito dall'amico **Silvio Savarino**, fondatore storico del gruppo di cracking **Hokuto Force** - http://csdb.dk/group/?id=435 e colgo l'occasione per ringraziarlo.

Invito chi fosse interessato a conoscere tale metodo a scrivere alla redazione di RetroMagazine all'indirizzo:

retromagazine.redazione@gmail.com

Cercheremo di accontentare tutti :)

Parte III – Le interruzioni di RASTER

La manipolazione del RASTER permette di ottenere molteplici risultati grafici. In questo articolo però, Vi illustrerò solo la parte strettamente utile per eseguire il brano nei giusti tempi. Per coloro che sono interessati a questo argomento, essendo molto tecnico e vasto, Vi rimando a successivi articoli e/o alla consultazione delle varie fonti in Internet.

Vi fornisco un link, tra i tanti, da cui iniziare:

http://codebase64.org/doku.php?id=magazines:chacking3#rasters_-_what_they_are_and_how_to_use_them

La conoscenza approfondita di tale argomento richiede parecchio impegno ma in compenso permette di ottenere molti effetti visivi straordinari utilizzabili per le intro o per i giochi.

Ad esempio è possibile ottenere barre di colore (statiche o in movimento) per tutta la lunghezza dello schermo, le cosiddette "rastersbar", il multiplexing degli sprites per poter visualizzare più di 8 sprites a video e molte altre cose ancora. Vi renderete conto già adesso, esaminando il codice che Vi presenterò a breve, della sua complessità, volendolo trattare in maniera ancora più approfondita.

Torniamo adesso al problema che ci siamo posti, ovvero la riproduzione di un brano .sid con la corretta temporizzazione. Dicevamo che lo schermo video viene continuamente ridisegnato (con una frequenza di 50 Hz,ovvero 50 volte al secondo, oppure di 60 Hz, 60 volte al secondo, in funzione del sistema video in uso) riga per riga, indipendentemente dal fatto che l'immagine riprodotta nel frattempo cambi o meno. Tale operazione viene effettuata da un cannone laser che percorre il reticolo di linee che compone lo schermo video (raster è il termine inglese che sta appunto per trama, reticolo, griglia). Il cannone laser eccita elettricamente il materiale di cui è composto lo schermo e vedremo quindi apparire, linea per linea, l'immagine sullo schermo.

Per dettagli tecnici ulteriori, rimando al link: https://it.wikipedia.org/wiki/Schermo_a_tubo_catodico

Il chip video VIC-II del C64 permette di gestire le interruzioni legate alla scansione delle linee di schermo. Arrivando a dover scansire una ben precisa linea di schermo, potremmo lanciare l'esecuzione di una routine che, tra le altre cose, si occuperà anche di eseguire il nostro brano .sid. Quanto detto è essenziale per comprendere il codice assembly allegato.

Parte IV – Commento relativo al codice assembly.

Il codice immediatamente dopo a: *=$0801 serve per generare l'istruzione SYS necessaria a lanciare in esecuzione il programma da BASIC. Il codice effettivo si trova a partire da *=$0810. Qui vengono inizializzati l'accumulatore, i registri X ed Y e viene richiamata la procedura di inizializzazione del file .sid. Con i tre valori o impostati, inizializzeremo la riproduzione del primo brano contenuto nel file .sid, spesso l'unico ma non è sempre così.
Subito dopo, mediante istruzione "SEI" vengono disabilitate le interruzioni IRQ. Vengono settati opportunamente i contenuti delle locazioni $dc0d e $dd0d (Interrupt control and status register).
Viene settato opportunamente il contenuto della locazione $d01A in maniera tale da abilitare le interruzioni di raster.

Per maggiore chiarezza ho espresso il valore da impostare in tali locazioni in notazione binaria, così da comprendere, bit per bit, il significato di tali valori. Viene impostato il valore delle locazioni $d011 (linea per esecuzione interruzione di raster), $d016 (Screen control register #2), $d018 (Memory setup register).
Viene modificato infine il vettore che punta alla routine di servizio IRQ reimpostandolo ad una nostra routine di servizio IRQ personalizzata. Eliminate eventuali interruzioni in sospeso, riabilitiamo le interruzioni con apposita istruzione "CLI" ed infine usciamo dalla esecuzione del programma con l'istruzione "RTS".
Quando il pennello ottico si troverà sulla riga di scansione che è stata impostata ($7E ovvero la 126-esima), partirà un segnale IRQ che interromperà il normale lavoro del microprocessore, verranno salvati i valori contenuti nei suoi registri sullo stack e verrà eseguita la routine indicata con la tag "irq" del programma di esempio.
In questa area di codice, viene mandato il segnale di "ACK" relativamente alla interruzione, evitando che la stessa interruzione venga nuovamente segnalata.
Successivamente, lanciamo la procedura di esecuzione del brano. Infine, eseguiamo la normale routine di servizio IRQ con la istruzione JMP $EA31. Notare le due istruzioni commentate con il simbolo di punto e virgola (;).
Se decommentate, togliendo il ; immediatamente alla loro sinistra, mostreranno una zona colorata nei bordi dello schermo video, in corrispondenza della linea scelta per l'interruzione di raster. Ho introdotto quelle due istruzioni, inc $d020 e dec $d020 (che incrementano e decrementano il valore di colore del bordo dello schermo) per darVi una idea di come vengano ottenute le cosiddette "rasterbars", cui Vi accennavo in precedenza.

Nella figura sopra il risultato di massima con le due istruzioni decommentate.

Infine, di seguito a *=$2000 viene caricato in memoria il file .sid preso ad esempio, a meno del suo header perchè inutile ai fini del suo impiego con CBM PRG STUDIO.
Il link da cui ho estratto le informazioni di massima relative alla memoria del C64 è il seguente:
http://sta.c64.org/cbm64mem.html

NOTA BENE
La tecnica descritta è generale, e andrà bene per la maggior parte dei casi. Tuttavia, ci sono files .sid che potrebbero necessitare di particolari accorgimenti per essere gestiti correttamente.
In particolare, occorre prestare particolare attenzione alle aree di memoria ove i files .sid vengono caricati ed a quali indirizzi si trovano le routine di inizializzazione e di riproduzione, per far sì che tali aree di memoria siano effettivamente accessibili nel caso in cui non lo siano di norma.
In molte circostanze potrà esserVi utile rilocare il file .sid in aree di RAM diverse da quelle previste originariamente.

Segnalo una ottima utility che permette di fare ciò: http://csdb.dk/release/?id=109000
Per maggiori dettagli, anche relativi alla utility segnalata, Vi rimando a futuri articoli.
Mi auguro che questo articolo sia stato di Vostro gradimento e di essere riuscito ad esporVi in maniera sufficientemente chiara, sintetica e completa quanto presentato.

In base anche al Vostro feedback, proveremo ad approfondire queste ed altre tematiche relative all'assembly del C64 e magari anche di altre piattaforme.

Programmazione Atari 2600 – bAtari Basic

di Giorgio Balestrieri

Programmare il 2600

Nel precedente numero di RetroMagazine abbiamo analizzato l'hardware del VCS per iniziare a prendere confidenza con la console e le sue capacità, prima di confrontarci con la produzione di software per questo sistema. In questo secondo articolo vedremo come un programmatore può utilizzare le caratteristiche della macchina, componendo dei piccoli demo per il VCS.

Chi ha letto lo scorso articolo, ricorderà che la programmazione di questa console richiede l'ottima conoscenza dell'hardware e notevoli capacità di sviluppo. Storicamente, il software prodotto per il 2600 è sempre stato scritto in linguaggio assembly ma, per nostra fortuna, la passione di chi ha amato questa console ha fatto nascere librerie di sviluppo e tool, tra cui il bAtari Basic, che semplificano molto il compito e con cui è possibile creare giochi anche complessi (per le possibilità della macchina) con molta meno fatica rispetto all'uso dell'assembly puro e liberandoci dall'onere di scrivere da zero ogni sua componente. In questo articolo si presuppone una conoscenza, anche basilare, del linguaggio Basic e la lettura e l'assimilazione delle nozioni contenute nell'articolo "Programmazione dell'Atari 2600 – L'hardware" pubblicato sul numero 4 di RetroMagazine, per cui vi consigliamo caldamente di (ri)leggerlo prima di procedere oltre e di tenerlo a portata di mano per una rapida consultazione.

Il bAtari Basic

Per la sua drastica scarsezza di supporto per l'hardware e di caratteristiche avanzate rispetto alle console concorrenti prodotte successivamente alla sua uscita, la 2600 è universalmente nota per essere una console stramaledettamente difficile da programmare. Ai programmatori è richiesto un enorme talento, una profonda conoscenza del sistema ed una notevole quantità di tempo ed energie per domarla ed imparare a sviluppare software, tutti fattori che scoraggiano i più visto che oggi, mancando quasi del tutto l'incentivo economico, l'unico compenso ottenibile è la soddisfazione personale.

A tutti gli aficionados con velleità di programmazione viene in aiuto il bAtari Basic, un framework di sviluppo che mette insieme un kernel (v. ***riq. 1***), interfacce software per la gestione dell'hardware ed un compilatore Basic dedicato che semplifica molto il compito dello sviluppatore. Non fatevi troppe illusioni però: anche con bAtari Basic (da qui in poi **bB**) scrivere software per il 2600 resta un compito non facile ma sicuramente possibile e con tempi di apprendimento decisamente ridotti.

In questo e nel prossimo articolo utilizzeremo Linux come piattaforma di sviluppo, ma è perfettamente possibile riprodurre tutti gli esempi ed applicare le nozioni qui acquisite tal quali su Windows, grazie alla natura multipiattaforma di **bB** e di Stella, l'emulatore di VCS che utilizzeremo per far girare tutti gli eseguibili prodotti dal compilatore. Per compilare gli esempi mostrati è necessario scaricare ed installare bAtari Basic, seguendo le instruzioni comprese nel pacchetto del tool. Per eseguire i file cartuccia prodotti da **bB**, è sufficiente installare Stella e darli in pasto all'emulatore come normali file rom. A proposito di Stella, il suo nome è derivato dal chip grafico della console, il TIA, ribattezzato dagli sviluppatori Stella. Per evitare di far confusione, qui ci riferiremo a TIA per indicare il chip ed a Stella per l'emulatore. Inoltre, poiché le capacità grafiche del 2600 variano a seconda del sistema NTSC, PAL o SECAM utilizzato, qui faremo riferimento al sistema NTSC che permette l'impiego della massima risoluzione video e numero di colori utilizzabili.

Struttura di bAtari Basic

Il pacchetto **bB** include un compilatore Basic che genera codice assembly compilabile da DASM, un assemblatore per CPU 65xx installato insieme a **bB** e diversi kernel (v. ***riq. 2***) per la gestione dello schermo, programmabili tramite istruzioni dedicate del Basic di **bB**. Il kernel standard provvede al disegno dello schermo utilizzando le capacità grafiche e sonore più o meno "lisce" della console, implementando un layout che prevede un campo di gioco ed un contatore a sei cifre disposto nella parte bassa dello schermo, utilizzabile come segnapunti. Specifiche istruzioni Basic sono dedicate alla gestione dei 5 sprite del TIA (player0, missile0, player1, missile1, ball), il playfield, di cui viene gestito anche lo scrolling, ed il colore dello sfondo. Altri kernel offrono capacità aggiuntive e possono essere attivati a seconda delle necessità del programmatore. In totale, **bB** fornisce ben 29 display kernel di cui 27 sono varianti dello standard kernel e due completamente differenti, il DPC+ ed il Multisprite, che permettono l'utilizzo di più sprite, anche multicolore e migliorie in diversi campi.

Per ragioni di spazio, in questo e nel prossimo articolo ci limiteremo all'utilizzo del kernel standard e del DPC+, quest'ultimo per le sue capacità grafiche avanzate, molto utili nello sviluppo di giochi.

Nei prossimi paragrafi esploreremo le caratteristiche del **bB** (versione 1.1d) e dello standard kernel, purtroppo non approfondendole come vorremmo per ragioni di spazio, ma abbastanza da poter riprodurre gli esempi presentati ed iniziare a sperimentare da soli, rimandando alla documentazione sul web coloro che volessero proseguire in questa avventura.

Struttura di un programma per Atari 2600

Il sorgente di un gioco prodotto con **bB** segue uno schema più o meno fisso. In testa è di solito presente una sezione di inizializzazione, dove vengono preparate le variabili da utilizzare nel corso del programma e la definizione degli elementi grafici. Segue poi il loop principale, dove si alternano una sezione di disegno del frame video ed una dove viene implementata la logica di gioco. A chiusura, una sezione di game over, che gestisce lo stato finale del programma e di norma prepara per il riavvio del gioco. Su questa struttura minimale possono essere innestate altre sezioni, ad esempio per la gestione di una schermata iniziale e/o di game over, sempre suddivise in una parte dedicata all'inizializzazione ed una al loop per il tracciamento di quanto mostrare a video.

Il Basic di bAtari Basic

La versione di Basic implementata da **bB** è piuttosto essenziale, modellata su quelle degli interpreti di prima generazione, anche se qui siamo affrancati dall'uso dei numeri di riga grazie alla supporto per le label utilizzabili come riferimenti per le istruzioni di salto e la possibilità di definire macro e funzioni per semplificare l'organizzazione e l'efficienza del codice. Ogni istruzione **deve** essere indentata di almeno uno spazio, ad eccezione delle label e del comando **end**. Le parole chiave e gli operatori debbono essere intervallati da uno spazio, per cui righe di questo tipo:

```
ifjoy0upthen_moveup=1:fora=1to4:p
fscrollup:next
```

che pure funzionano su molto interpreti basic, sono illeggibili tanto per un umano quanto per il compilatore, e vanno scritte come:

```
if joy0up then _moveup = 1: for a
= 1 to 4: pfscroll up: next
```

Gli unici tipi di dati utilizzabili sono gli interi ad 8 bit ed i decimali a virgola fissa con 8 o 4 bit per la parte intera ed 8 o 4 per quella decimale. Non sono previsti tipi stringa, il che è abbastanza comprensibile visto che il VCS non ha la gestione dei caratteri e non può stampare messaggi a video (debbono essere simulati con gli sprite o il playfield). E' possibile però definire blocchi di dati per la composizione degli elementi grafici o generici array di valori.

Il set di istruzioni può essere raggruppato in sezioni a seconda dell'utilizzo per cui sono previste. La suddivisione non è rigida e normalmente varia a seconda dell'autore della documentazione, libro o articolo sul tema. Qui le abbiamo classificate in quattro blocchi, mostrati nelle ***tabelle 1, 2, 3*** e ***4***.

In ***tabella 1*** troviamo le istruzioni generiche, ovvero tutte quelle per la scrittura di codice non specifico per la console. In questo gruppo troviamo istruzioni per la definizione delle variabili, gestione del flusso del programma, cicli ecc.

In ***tabella 2*** troviamo le istruzioni dedicate alla gestione delle capacità grafiche del TIA implementate dallo standard kernel, di cui molte comuni anche al DPC+ ed al Multisprite kernel, ed agli input provenienti dai sei switch posizionati sulla console e dalle due porte controller. Di norma, queste istruzioni vengono utilizzate nel ciclo principale del gioco.

Figura 1 – Hello World

In ***tabella 3*** sono contenute le istruzioni per agire direttamente sui registri del TIA.

La ***tabella 4*** raggruppa le istruzioni specifiche del kernel DPC+ e di altri che vengono comunemente utilizzati per la scrittura di giochi in **bB**.

In questa sede, causa limiti di spazio, non descriveremo l'intero set di istruzioni né, purtroppo, terremo un corso di linguaggio Basic, ma ci limiteremo a procedere per esempi, descrivendo di volta in volta le istruzioni utilizzate, rimandando alla documentazione sul web gli eventuali approfondimenti. Anche se qui è una necessità, questo modo di procedere ha il vantaggio di permettere l'applicazione immediata delle nozioni apprese e di annoiare meno il lettore. O almeno, lo speriamo.

Hello World

Tenendo d'occhio le tabelle delle istruzioni di **bB** e ricordando come funziona il ciclo software dell'Atari 2600, siamo pronti per scrivere il primo programma in bAtari Basic. Nel rispetto della tradizione, il nostro esempio si limiterà a mostrare sullo schermo il messaggio "Hello World", con la variante di cambiare il colore in cui è stampato ad ogni quadro video.

Il codice necessario per questo semplice programma lo si può vedere nel ***listato 1***; per compilarlo è sufficiente copiarlo in un file di testo con estensione .bas e darlo in pasto a **bB** con il comando **2600basic.sh** su Linux o **2600basic.bat** su Windows. Verranno prodotti una serie di file, di cui uno con estensione .bin che potremo far eseguire dall'emulatore Stella. L'output è visibile in ***fig. 1***, ovviamente senza l'effetto dei colori cangianti.

Pur nella sua semplicità, questo piccola dimostrazione ha molto da insegnarci. Il programma prevede due sole sezioni, una di definizione delle risorse necessarie ed una di core, che fa effettivamente "girare" il programma e disegna i frame video. Alla riga 1 troviamo la dichiarazione di un blocco di dati, **playfield:**, che definisce la struttura grafica dell'oggetto playfield del TIA. Come ricorderete dallo scorso articolo, il campo da gioco è un elemento statico (in teoria almeno) deputato al disegno del fondale di gioco. Lo standard kernel utilizza di default una dimensione di 32x11 pixel per questo oggetto e gli dona la capacità di movimento, così da poterlo posizionare arbitrariamente sullo schermo o farlo scorrere (in gergo tecnico si usa il neologismo "scrollare") nelle otto direzioni cardinali tramite le istruzioni **playfieldpos** e **pfscroll**. Se vi state chiedendo come sia possibile con una dimensione orizzontale di 32 pixel ottenere l'effetto di riempire lo schermo per la sua intera lunghezza è presto detto. Nel VCS la dimensione dei pixel non è definita ed è data dal tempo (color clock) che lasciamo al TIA per disegnarne uno. Lo standard kernel utilizza di default pixel da 8 color clock, il che spiega l'arcano. Tornando all'istruzione **playfield:**, vediamo che le righe successive disegnano una matrice composta da caratteri "." ed "X". Un "." indica un pixel spento, mentre la "X" uno acceso; in questo modo è possibile modellare facilmente l'aspetto di playfield. L'instruzione **end** termina **playfield:**. Notare l'assenza di indentazione che **deve** essere presente per tutte le altre istruzioni ed **assente** per le label e l'istruzione **end**.

Alla riga 15, troviamo una istruzione **dim** che, contrariamente a quanto avviene nel Basic standard, qui non serve a definire array o assegnare un tipo di dati ad una variabile, bensì a creare un alias di una delle variabili messe a disposizione dal linguaggio. A differenza dei Basic a cui si è forse abituati, in **bB** non è possibile creare variabili. A seconda del kernel utilizzato, abbiamo un numero fisso di variabili da poter sfruttare, suddivise in stabili, che conservano il loro valore per tutto il ciclo del programma (a meno che non lo cambiamo) e temporanee, il cui valore è effimero e viene perso dopo il disegno di un frame. Con lo standard kernel abbiamo 26 variabili stabili, contrassegnate da una lettera dell'alfabeto (dalla **a** alla **z**) e 7 temporanee, da **temp1** a **temp7**. In teoria almeno; in pratica, poiché **temp7** viene utilizzata nelle operazioni di bankswitching (v. ***riquadro 2***), è estremamente saggio evitare di impiegarla, a meno che non ci si voglia creare problemi piuttosto seri. L'istruzione **dim** in **bB** viene dunque semplicemente utilizzata per assegnare un nome meno criptico ad una variabile. Nel nostro esempio, alla variabile **a** viene assegnato l'alias **_PlayFieldColor** che ricorda lo scopo a cui è destinata.

Procedendo con la lettura del codice, alla riga 17 troviamo la definizione di una label che coincide con il ciclo principale del programma, quello in cui viene disegnato un frame video ed implementata la logica del nostro gioco. Riferendoci allo schema dei tempi e delle modalità di composizione di un frame visto nello scorso articolo, ricordiamo che il codice non dedicato al disegno del quadro video può girare solo durante la fase di VSYNC e quella di OVERSCAN. In **bB**, a meno di utilizzare l'istruzione **vsync**, il codice per il gioco gira solo durante l'OVERSCAN. Alla riga 18 viene assegnato un colore a **playfield** utilizzando la variabile dedicata **COLUPF** a sola scrittura che mappa il corrispondente registro del TIA mentre alla riga 19, tramite la variabile di sistema **scorecolor**, assegnamo un colore al segnapunti, sottraendo a 255 il colore del playfield. Poichè tutte le variabili vengono inizializzate al valore zero dal **bB** all'avvio del programma, qui non ci siamo preoccupati di impostarne un valore di inizio. Questa è però un'eccezione; in un gioco vero, che può essere riavviato tramite lo switch reset sulla console, è importantissimo assegnare un valore alle variabili che verranno poi utilizzate nel corso del programma, anche quelle che contengono zero.

La riga 21 contiene un'istruzione molto importante: **drawscreen**. Con essa passiamo il controllo al kernel in uso (lo standard kernel in questo caso) per il disegno del frame video. Gran parte della magia di **bB** avviene qui. Disegnare un frame video è un'operazione tutt'altro che banale e farlo "a mano", scrivendo da zero un nostro kernel che raggiunga le performance dei kernel di bB è un'operazione che richiede grandi abilità e una sfida niente affatto facile. Per chi volesse cimentarsi nell'impresa, **bB** offre il supporto ai custom kernel, da poter poi integrare nel tool.

Figura 2 – Joystick e sprite

Dopo che il quadro video è stato disegnato, il controllo ritorna al nostro programma per l'esecuzione del codice che ne rappresenta la logica. Il tempo che abbiamo a disposizione per effettuare le operazioni necessarie allo svolgimento del gioco, come il calcolo della posizione degli sprite, controllo delle collisioni, reazioni agli input ecc., è meno di 2 millisecondi, per cui l'ottimizzazione del codice è d'obbligo anche con il bAtari Basic. Il mancato rispetto di questo vincolo porta ad una schermata piena di artefatti, tremolante o che addirittura inizia a scorrere come in un televisore mal sintonizzato.

Alla riga 23, tramite la variabile riservata **score**, incrementiamo il segnapunti di un'unità mentre alla riga successiva calcoliamo il nuovo colore del playfield incrementando il valore della variabile che lo contiene di 1. I colori disponibili nella tavolozza NTSC sono 128, numerati da 0 a 127 (v. ***figura 4***) ma si ripetono da 128 a 255 per cui, assegnare un colore al di fuori del range 0-127, funziona comunque. Inoltre, poichè le variabili in **bB** sono tutte di tipo intero ad 8 bit possono solo contenere valori nell'intervallo 0-255. L'assegnazione tramite espressione di valori maggiori farà scattare l'overflow ed il valore verrà riportato in quel range automaticamente (es. 256 diventa 0). L'unica variabile che non segue questa regola è **score**, che può avere valori compresi tra 0 e 999999, codificati in BCD. L'effetto pratico di queste considerazioni è che il sistema utilizzato per cambiare i colori al playfield ed al segnapunti, seppur non rigorosamente corretto, da il risultato desiderato, liberandoci dall'obbligo di testare il valore delle variabili ad ogni giro.

L'ultima istruzione, alla riga 26, è un'istruzione di salto che rimanda alla label **mainloop** per ricominciare il ciclo e ridisegnare il quadro video con le nuove impostazioni di colore e valore di **score**.

Prima di passare al prossimo paragrafo, in cui mostreremo come leggere l'input di un joystick e muovere uno sprite sullo schermo, è buona cosa impiegare un po' di tempo a sperimentare con questo esempio ed effettuare piccole modifiche così da prendere confidenza con il linguaggio e probabilmente, scoprire alcune delle singolarità di questo tool. Come esercizio, provate a nascondere la linea dello score, assegnandogli il colore dello sfondo oppure disabilitandolo tramite l'uso dell'istruzione dedicata a questo scopo.

Joystick e sprite

Passiamo ora ad un esempio un po' più complesso ma anche più interessante, dove vedremo l'uso degli sprite e la gestione dei joystick. Il codice è contenuto nel ***listato 2***; se compilato e fatto girare, seguendo la stessa procedura di Hello World, mostra un riquadro intorno allo schermo, ottenuto tramite l'impiego del playfield ed uno sprite, che rappresenta un disco volante stilizzato, al centro dello video (***fig. 2***). Muovendo il joystick collegato alla porta 1 si può spostare il disco volante su tutto lo schermo, restando nei limiti del perimetro tracciato dal playfield, le cui coordinate sono indicate dalle quattro costanti definite nelle righe da 1 a 4. Alla riga 6, la costante riservata **scorefade** viene impostata ad 1 per attivare la modalità multicolore del segnapunti e nella riga successiva ne viene indicato il colore di base che verrà automaticamente sfumato per tutte le cifre dello score. Notare che invece di utilizzare un numero decimale, in questo caso abbiamo utilizzato il valore esadecimale **$B8** per l'assegnazione. Poiché per le variabili di **bB** possono essere utilizzati indifferentemente valori decimali, esadecimali o binari e dal momento che la tabella dei colori (***fig. 4***) ha i riferimenti in esadecimale, per semplicità abbiamo usato questo formato per scegliere il colore desiderato. Alle righe 9 e 10 troviamo ancora due variabili di sistema, **player0x** e **player0y**, deputate al posizionamento dello sprite playero che avviene durante l'esecuzione di **drawscreen**. Da notare che questo è un'altra caratteristica in aiuto allo sviluppatore fornitaci dallo standard kernel. Il TIA infatti non prevede un sistema a coodinate per posizionare gli sprite; per mostrarli in una data posizione, occorre attivarne la visualizzazione nel punto dello schermo in cui si vuole che appaiano e spostarli con comandi di posizionamento relativo del tipo "sposta playero a destra di 1 color clock" ecc. Grazie allo standard kernel (ma anche al Multisprite ed al DPC+ kernel), possiamo gestirli molto più facilmente indicando le loro posizioni X ed Y sullo schermo. Le righe 12 e 13 assegnano il colore al playfield ed allo sfondo tramite le variabili **COLUPF** e **COLUBK**, che mappano i rispettivi registri del TIA. Continuando l'analisi del codice, troviamo la definizione dell'aspetto grafico dello sprite playero tramite il blocco di dati **player0** dedicato a questo scopo. La sua composizione è simile a quella vista per il playfield ma presenta alcune differenze. La prima che salta all'occhio, è che qui dobbiamo utilizzare la rappresentazione binaria (un numero binario viene indicato dall'uso del simbolo '%') per definire un numero ad 8 bit (la larghezza massima). L'altezza dello sprite è decisa dal programmatore, indicando semplicemente tante righe quante ne occorrono (fino ad un massimo di 256) ma debbono dichiarate in ordine inverso rispetto a come dev'essere disegnata l'immagine (v. ***fig. 5***). Segue poi la definizione del playfield, che in questo caso consiste in un semplice un bordo intorno tracciato intorno allo schermo. L'istruzione alla riga 40 attiva lo smart

branching, che permette la gestione corretta dei salti all'interno dei blocchi di **if**. Anche se in questo demo non vengono utilizzati **goto** negli **if**, è bene abituarci da subito ad utilizzarlo ed evitarci eventuali errori "branch out of range" da parte del compilatore in futuro. Il costo dell'uso dello smartbranching è la produzione di un codice assembly più difficile da leggere, ma se non avete intenzione di manipolare l'output intermedio di **bB** potete tranquillamente ignorare questo aspetto; la velocità di esecuzione del codice resta la stessa.

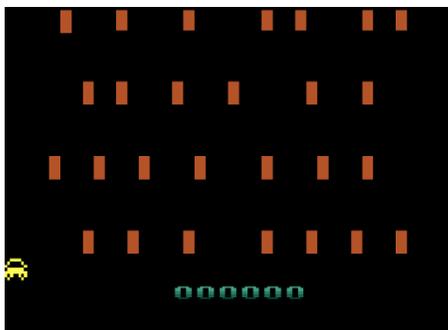



Figura 3 – Playfield scrolling

Alla riga 42 inizia il loop principale del programma ed alla 44, tramite la variabile mappata del TIA, viene indicato un colore per lo sprite player0. A differenza delle variabili per il background ed il playfield, questa è una variabile effimera, il cui valore viene perso dopo il disegno di un quadro video. Per questo motivo va utilizzata all'interno del loop principale, pena il reset del colore dello sprite dopo il primo frame. Le righe da 48 a 51 contengono i controlli per la lettura del joystick ed il relativo spostamento dello sprite a video. Lo stato del joystick viene riportato nelle funzioni di sistema **joy0up**, **joy0down**, **joy0left**, **joy0right** e **joy0fire** che valgono **true** se la leva del controller viene spinta nella direzione corrispondente o se viene premuto l'unico bottone disponibile. Quando viene rilevata una direzione indicata dal joystick, le coordinate X o Y dello sprite (**player0x** e **player0y**) vengono incrementate o diminuite in accordo, previo il controllo della posizione per accertarci che non si sia già raggiunto il bordo definito dal playfield. Una volta impostate le coordinate dello sprite, esse avranno effetto alla chiamata di **drawscreen**. Notare l'uso dell'operatore booleano **&&** invece dell'**and** utilizzato in altri dialetti Basic. A proposito degli operatori booleani **and** (**&&**) e **or** (**||**) negli **if**, in **bB** esistono un po' di limiti sul loro utilizzo: possiamo utilizzare quanti **and** vogliamo in un **if** ma è permesso un solo **or** e non è consentito l'uso di **and** e **or** nella stessa espressione.

Ultima istruzione del programma, alla riga 53, il **goto** per far ricominciare il loop.

Nel prossimo paragrafo, vedremo un terzo ed ultimo esempio di programma in cui combineremo la gestione di sprite, joystick e scrolling del playfield insieme. Se avete voglia, prima di passare oltre provate a modificare questo esempio per visualizzare due sprite contemporaneamente ad esempio (utilizzare cioè anche player1) o a limitare il movimento a 4 direzioni invece che 8 come ora (bastano 4 **goto** ed una label).

Scrolling del playfield

Grazie ai due esempi visti nei paragrafi precedenti, ora sappiamo come aggiungere elementi grafici al campo di gioco, disegnare sprite, spostarli sullo schermo e leggere l'input dei joystick. In questo ultimo esempio, riutilizzeremo tutto questo aggiungendo lo scrolling del playfield, caratteristica non prevista dal TIA ma possibile grazie a particolari tecniche di programmazione che, ancora una volta, lo standard kernel mette in campo per noi. Vedremo anche come gestire le collisioni tra sprite e fondale, grazie ad una semplice funzione di **bB** che al solito gestisce il TIA per noi. Il ***listato 3***, che potete copiare in un file e compilare come al solito, realizza un piccolo gioco in cui lo scopo è portare un disco volante dall'angolo in basso a sinistra dello schermo in quello in alto a destra, evitando di colpire i blocchi rettangolari che si muovono secondo uno schema fisso, pena il ritorno alla posizione iniziale. Una volta raggiunto l'angolo in alto a destra, viene assegnato un punto e il disco volante viene rispedito nell'angolo in basso a sinistra per ricominciare il ciclo. Vediamo in dettaglio come tutto ciò viene realizzato.

Le prime quattro righe impostano delle costanti che rappresentano i confini dello schermo; queste verranno controllate ad ogni movimento dello sprite player0 per evitare che il disco esca dal quadro visibile. Alle righe 5 e 6 troviamo altre due costanti che contengono la posizione, in coordinate X ed Y, del punto di partenza dello sprite. Alla riga 8 viene definito un alias, **_FrameCounter**, che mantiene un contatore di frame. Più avanti questa variabile verrà utilizzata per decidere ogni quanti frame far scorrere il playfield, decidendone di fatto la velocità di movimento. La variabile **_DirChange** (o meglio, l'alias) alla riga 9 definisce un contatore utilizzato per decidere quando cambiare direzione allo scorrimento del playfield. Sulla riga successiva viene creato l'alias **_Dir** che contiene un valore da 0 a 3 utilizzato per indicare in che direzione far muovere il playfield. Le istruzioni dalla riga 12 alla riga 44 dovrebbero essere ormai note; in breve, definiscono la posizione iniziale di player0, i colori di playfield e sfondo e la composizione dello sprite del giocatore 1 e del fondale, composto da sporadici pixel accesi qua e là, la cui funzione è ostacolare il disco volante nelle sue peregrinazioni. Alla riga 46, attiviamo lo smartbranching per i motivi spiegati precedentemente.

La riga 48 da inizio al loop principale di gioco, anche in questo caso composto da due sole fasi, una di disegno del frame video ed una di logica di gioco. Viene assegnato il colore allo sprite player0 (riga 50) e dato il via allo standard kernel per disegnare lo schermo (riga 52), passando poi alla gestione dello spostamento del disco volante interrogando lo stato del joystick, in maniera del tutto analoga a quanto implementato nell'esempio precedente.

Alla riga 59, viene incrementato di una unità il contatore di frame **_FrameCounter** che viene utilizzato nei quattro **if** successivi per decidere se far muovere o meno il playfield. Se **_FrameCounter** vale 6 (cioè sono stati disegnati sei frame video), allora playfield viene spostato di un pixel nella direzione indicata da **_Dir**. Se **_Dir** vale 0, viene spostato verso il basso, se vale 1, verso l'alto, 2 e 3 lo muovono rispettivamente a sinistra e destra. In ogni caso, una volta fatto scrollare playfield (istruzioni **pfscroll**), **_FrameCounter** viene azzerato e la variabile **_DirChange** incrementata di 1.

Alla riga 66, **_DirChange** viene riportata a zero se ha raggiunto o superato il valore di 5 e **_Dir** incrementata di una unità. L'effetto pratico è quello di far cambiare direzione al playfield dopo avergli fatto percorrere 5 pixel. La riga 67 azzera **_Dir** se il suo valore ha raggiunto le quattro unità, visto che i valori a noi utili sono solo quelli da zero a tre

Alla riga successiva c'è il controllo della collisione tra player0 e playfield, utilizzando l'istruzione **collision**, che ritorna **true** se c'è collisione, **false** altrimenti. Fornendo a **collision** coppie diverse di elementi grafici, è possibile testare l'impatto tra i vari oggetti del TIA, cioè la condizione in cui il pixel di uno sprite si sovrappone a quello di un altro o del playfield.

Il blocco **if** alla riga 71 controlla che il disco volante sia giunto sano e salvo nell'angolo in alto a destra dello schermo, condizione in cui viene riconosciuto un punto al giocatore ed il disco riportato alla posizione iniziale, utilizzando istruzioni già analizzate negli esempi precedenti.

L'ultima istruzione, come da prassi, è quella di salto per tornare a `mainloop` e ricominciare il ciclo.

Dal punto di vista didattico, questo esempio fornisce grande libertà di sperimentazione, grazie al numero di caratteristiche di **bB** utilizzate ed alla sua completezza, pur nella sua minimalità. A titolo di esercizio, suggeriamo di provare a:

- Utilizzare lo sprite player1 e di posizionarlo nell'angolo in alto a destra per rappresentare visivamente l'obiettivo da raggiungere.
- Utilizzare lo sprite player1 e di posizionarlo in un punto dello schermo, magari casuale (utilizzare l'istruzione `rand`), assegnando 5 punti al giocatore se riesce a raggiungerlo. Una volta raggiunto, l'oggetto bonus va rimosso e riposizionato sullo schermo solo dopo che il disco ha completato con successo la sua missione, ovvero è arrivato nell'angolo in alto a destra.
- Cambiare la velocità di scorrimento del playfield e l'ordine della sequenza di movimento.
- Cambiare la durata del movimento del playfield in una data direzione, provando anche a definire durate diverse per direzioni diverse (variare cioè il numero di pixel percorsi dal playfield in una direzione assegnata prima di cambiarla).
- Utilizzare una costante invece del valore assoluto per il numero di frame da attendere prima di muovere il playfield.
- Utilizzare delle costanti invece dei valori assoluti per il numero di frame durante i quali il playfield viene spostato in una data direzione.

Date un'occhiata alla documentazione disponibile sul web per ulteriori approfondimenti e spunti e, se volete, condividete con noi i risultati raggiunti scrivendo una mail alla redazione.

Conclusioni

Termina qui questo breve, ma speriamo sufficientemente interessante, tour nel bAtari Basic, di cui abbiamo visto un numero sufficiente di caratteristiche da poter iniziare a sviluppare giochi per questa straordinaria piattaforma. Ci auguriamo che alla vostra curiosità siano stati forniti stimoli sufficienti per continuare ad approfondire **bB** da soli, naturalmente con l'ausilio della documentazione on-line.

Nel prossimo e conclusivo articolo di questa serie, presenteremo ed analizzeremo un piccolo gioco sviluppato da noi, così da confrontarci con un software completo e continuare l'analisi di **bB**, utilizzando altre utili ed importanti caratteristiche di questo ottimo, seppur incompleto, strumento di sviluppo.

Per chi volesse provare gli esempi mostrati in questa sede o lanciarsi nello studio di bAtari Basic, consigliamo i seguenti link:

- bAtari Basic, reperibile all'url:
  - http://7800.8bitdev.org/index.php/Batari_basic
- Stella, l'emulatore per eccellenza di Atari 2600:
  - https://stella-emu.github.io/ (per chi usa Linux, probabilmente lo trova già nel software repository della propria distribuzione).
- Documentazione di bAtari Basic:
  - http://www.randomterrain.com/atari-2600-memories-batari-basic-commands.html
  - https://alienbill.com/2600/basic/downloads/0.35/help.html
- Raccolta di programmi sviluppati in **bB** (purtroppo solo binari):
  - http://www.atari2600land.com/basic/index.html

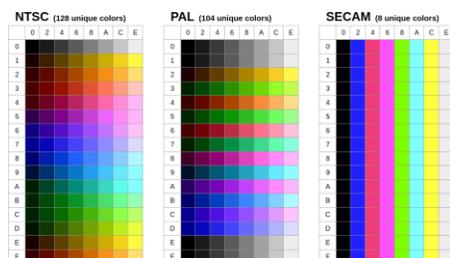



Figura 4 - Palette Atari 2600

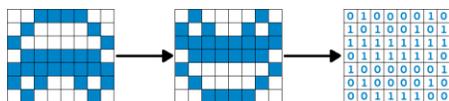



Figura 5 – Disegno sprite

In senso generico, un kernel è il nucleo di un sistema operativo deputato ad assolvere funzioni di base e vitali per il funzionamento del sistema stesso. Nell'Atari 2600, che non disponeva di un sistema operativo, con kernel si indicava la parte di programma che aveva il compito di gestire l'hardware, ed in particolare di disegnare i frame video. I kernel più efficienti, prodotti da programmatori leggendari come David Crane, hanno permesso la nascita di giochi come Pitfall II o Solaris (di Doug Neubauer). Il bAtari Basic include diversi kernel, tra cui uno ispirato al chip DPC progettato da Crane per Pitfall II, che sollevano il programmatore dall'ardua impresa di svilupparne uno ex-novo, compito comunque possibile e previsto dal tool di sviluppo.

Riquadro 1 – Kernel

Nei vecchi processori ad 8bit, per superare i limiti sulla quantità di memoria a cui la CPU poteva accedere, si utilizzava una tecnica nota come bankswitching. In pratica, la memoria totale veniva suddivisa in blocchi, detti banchi, che venivano "mostrati" al processore agendo su una cirtuiteria che collegava al bus di indirizzi del processore un banco di memoria scelto dal programmatore. Il MOS 6502, con un bus di indirizzi a 16bit era in grado di "vedere" una finestra di 64k di memoria ma il 6507, versione ridotta del 6502 ed utilizzato nel VCS, poteva accedere ad uno spazio di indirizzi di soli 4k. Grazie al bankswitching, il 2600 poteva utilizzare quantità più grandi di memoria, a vantaggio della qualità dei giochi. Il bAtari Basic fornisce opzioni per gestire il bankswitching facilmente e permette di definire la dimensione della rom che conterrà il nostro programma con una semplice direttiva per il compilatore.

Riquadro 2 - Bankswitching

`asm`	Definizione blocco di codice in assembly.
`bank, romsize`	Gestione banchi di memoria e dimensione della rom da utilizzare.
`const`	Imposta una costante.
`dec`	Espressioni matematiche decimali (BCD).
`dim, def`	Rinomina una variabile o definisce un'espressione.
`end`	Fine di un blocco dati o assembler.
`for … step … next`	Ciclo di istruzioni.
`function`	Definizione di funzioni.
`gosub … return`	Salta ad un altro punto del programma e riprende l'esecuzione all'istruzione successiva al salto.
`goto`	Salta ad un altro punto del programma.
`if … then … else`	Controllo condizionale del flusso di programma.
`include, inline`	Inclusione di moduli in assembly.
`includesfile`	Inclusione di file basic.
`let`	Assegnazione di un valore ad una variabile (deprecato).
`macro, callmacro`	Definizione e chiamata di macro.
`on … goto, on … gosub`	Salto su condizione.
`pop`	Annulla il punto di ritorno dell'ultimo gosub.
`rand`	Genera numeri pseudo casuali.
`read, data … end`	Lettura e definizione di un blocco arbitrario di dati.
`Rem, “;”, /*---*/`	Commento su riga singola o su più righe.
`return thisbank, return otherbank`	Ritorno da un gosub con gestione banchi di memoria.
`sdata, sread … end`	Lettura e definizione di un blocco di dati sequenziali-
`set`	Impostazioni di direttive per il compilatore.

Tabella 1 – Istruzioni generiche

`ballx, bally, ballheight`	Coordinate e dimensione dello sprite ball.
`const noscore`	Mostra o nasconde il segnapunti.
`const pfres`	Numero di righe di playfield.
`const pfrowheight`	Altezza dei "pixel" del playfield.
`const pfscore`	Abilita o disabilita due segnapunti separati.
`drawscreen`	Esegue il kernel e disegna lo schermo.
`joy0up, joy0down, joy0left, joy0right, joy0fire`	Stato del joysticko.
`joy1up, joy1down, joy1left, joy1right, joy1fire`	Stato del joystick1.
`missile0x, missile0y, missile0height`	Coordinate e dimensione dello sprite missileo.
`missile1x, missile1y, missile1height`	Coordinate e dimensione dello sprite missile1.
`pfclear, pfhline, pfvline`	Controllo del playfield.
`pfread, pfpixel`	Legge o modifica un pixel del playfield.
`pfscore1, pfscore2`	Valore (in binario) dei segnapunti.
`pfscorecolor`	Colore dei segnapunti.
`pfscroll, playfieldpos`	Effettua lo scrolling e riposiziona il playfield.
`player0height, player1height`	Modifica l'altezza degli sprite player o ed 1 dinamicamente.
`player0:, player1:`	Definizione degli sprite playero e player1.
`player0x, player0y`	Coordinate dello sprite playero.
`player1x, player1y`	Coordinate dello sprite player1.
`playfield:`	Definizione del blocco grafico di playfield.
`readpaddle, currentpaddle`	Stato del controller racchetta.
`reboot`	Reset a caldo della cartuccia.
`score`	Valore del segnapunti (in BCD).
`scorecolor`	Colore del segnapunti.
`switchreset, switchbw, switchselect, switchleftb, switchrightb`	Stato degli interruttori della console.
`vblank`	Definisce il codice da far girare nel periodo di vblank. Da non usare con il kernel DPC+.

Tabella 2 – Istruzioni la gestione dello schermo e degli input.

AUDV0, AUDC0, AUDF0	Audio 0.
AUDV1, AUDC1, AUDF1	Audio 1.
COLUBK	Colore dello sfondo.
COLUP0, COLUP1	Colore di player0 e player1 (hanno effetto anche su missile0 e missile1).
COLUPF	Colore di playfield (ha effetto anche su ball).
CTRLPF	Proprietà di playfield e ball (inversione speculare, priorità rispetto a player0 e player1, attribuzione colore delle due metà, dimensione di ball).
NUSIZ0, NUSIZ1	Proprietà degli sprite player e missile (dimensione e numero di copie).
PF0	Accesso diretto al buffer grafico di playfield.
REFP0, REFP1	Inverte specularmente gli sprite player0 e player1.

Tabella 3 – Registri del TIA (Stella)

DPC+ kernel	
set dpcspritemax #	Limita il numero degli sprite emulati (libera memoria).
player0color:, player1color:	Colori player0 e player1, un colore per riga.
stack, pull, push	Istruzioni per la gestione dello stack.
scorecolors:	Colori per il segnapunti, un colore per riga.
Life Counter e Status Bar Loop kernel	
inline 6lives_statusbar.asm	Carica il kernel Status Bar.
inline 6lives.asm	Carica il kernel Life Counter, per mostrare il numero di vite rimanenti (icona e numero).
lives	Indicatore vite rimaste (icona e numero).
lives:	Blocco definizione icona vite.
lifecolor	Colore icona e numero vite.
statusbarlength	Lunghezza barra di stato.
statusbarcolor	Colore barra di stato.

Tabella 4 – Istruzioni dedicate a kernel non standard

```
01     playfield:
02     .XXX.XXX.XXXXX.XXX...XXX....XX..
03     ..X...X..X...X..X.....X....X..X.
04     ..XXXXX..XXX....X.....X....X..X.
05     ..X...X..X...X..X..X..X..X.X..X.
06     .XXX.XXX.XXXXX.XXXXX.XXXXX..XX..
07     ................................
08     XXX.XXX..XX..XXXX...XXX...XXXX..
09     .X...X..X..X..X..X...X.....X..X.
10     .X.X.X..X..X..XXX....X.....X..X.
11     .X.X.X..X..X..X..X...X..X..X..X.
12     ..X.X....XX..XXX..X.XXXXX.XXXX..
13   end
14
15     dim _PlayFieldColor = a
16
17   mainloop
18     COLUPF = _PlayFieldColor
19     scorecolor = 255 - _PlayFieldColor
20
21     drawscreen
22
23     score = score + 1
24     _PlayFieldColor = _PlayFieldColor + 1
25
26     goto mainloop
```

Listato 1 – Hello World

```
01    const _Top     = 15
02    const _Bottom  = 80
03    const _Left    = 21
04    const _Right   = 133
05
06    const scorefade = 1
07    scorecolor = $B8
08
09    player0x = 77
10    player0y = 53
11
12    COLUPF = $34
13    COLUBK = 0
14
15    player0:
16    %01000010
17    %10100101
18    %11111111
19    %01111110
20    %10000001
21    %01000010
22    %00111100
23  end
25
26    playfield:
27    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
28    X..............................X
29    X..............................X
30    X..............................X
31    X..............................X
32    X..............................X
33    X..............................X
34    X..............................X
35    X..............................X
36    X..............................X
37    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
38  End
```

Listato 2 – Joystick e sprite

```
39
40    set smartbranching on
41
42  mainloop
43
44    COLUP0 = $1E
45
46    drawscreen
47
48    if joy0up     && player0y > _Top    then player0y = player0y - 1
49    if joy0down   && player0y < _Bottom then player0y = player0y + 1
50    if joy0left   && player0x > _Left   then player0x = player0x - 1
51    if joy0right  && player0x < _Right  then player0x = player0x + 1
52
53    goto mainloop
```

Listato 2 – Joystick e sprite (continua)

```
00    const _Top      = 8
01    const _Bottom   = 88
02    const _Left     = 1
03    const _Right    = 153
04    const _HomeX    = 1
05    const _HomeY    = 88
06
07    dim _FrameCounter = a
08    dim _DirChange    = b
09    dim _Dir          = c
10
11    const scorefade = 1
12    scorecolor = $B8
13
14    player0x = _HomeX
15    player0y = _HomeY
16
17    COLUPF = $34
18    COLUBK = 0
19
20    player0:
21    %01000010
22    %10100101
23    %11111111
24    %01111110
25    %10000001
26    %01000010
27    %00111100
28   end
29
30
31    playfield:
32    .X....X.....X......X..X.....X..X
33    ................................
34    ................................
35    ...X..X....X....X......X....X...
36    ................................
37    ................................
38   X...X...X....X.....X....X...X...
39    ................................
```

Listato 3 – Playfield e scrolling

```
40    ................................
41    ...X...X....X......X...X...X...X
42    ................................
43  end
44
45    set smartbranching on
46
47  mainloop
48
49    COLUP0 = $1E
50
51    drawscreen
52
53    if joy0up     && player0y > _Top    then player0y = player0y - 1
54    if joy0down   && player0y < _Bottom then player0y = player0y + 1
55    if joy0left   && player0x > _Left   then player0x = player0x - 1
56    if joy0right  && player0x < _Right  then player0x = player0x + 1
57
58    _FrameCounter = _FrameCounter + 1
59
60    if _FrameCounter = 6 && _Dir = 0 then pfscroll down:  _FrameCounter = 0: _DirChange = _DirChange + 1
61    if _FrameCounter = 6 && _Dir = 1 then pfscroll up:    _FrameCounter = 0: _DirChange = _DirChange + 1
62    if _FrameCounter = 6 && _Dir = 2 then pfscroll left:  _FrameCounter = 0: _DirChange = _DirChange + 1
63    if _FrameCounter = 6 && _Dir = 3 then pfscroll right: _FrameCounter = 0: _DirChange = _DirChange + 1
64
65    if _DirChange >= 5 then _Dir = _Dir + 1: _DirChange = 0
66    if _Dir = 4 then _Dir = 0
67
68    if collision(player0, playfield) then player0x = _HomeX: player0y = _HomeY
69
71    if player0x = _Right && player0y = _Top then  player0x = _HomeX: player0y = _HomeY: score = score + 1
72
73    goto mainloop
```

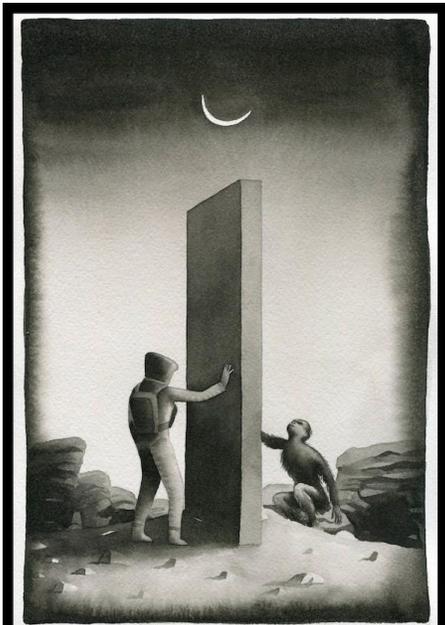
Listato 3 – Playfield e scrolling (continua)

Come scoprii i giochi RPG con Il TI99/4A

Ovvero come la scimmia trovò la strada dell'evoluzione...

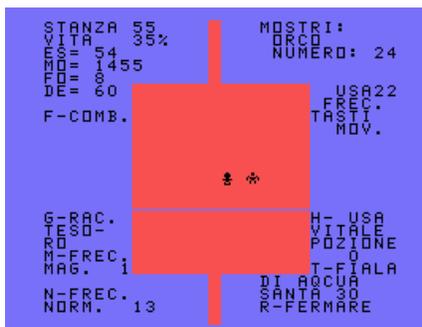
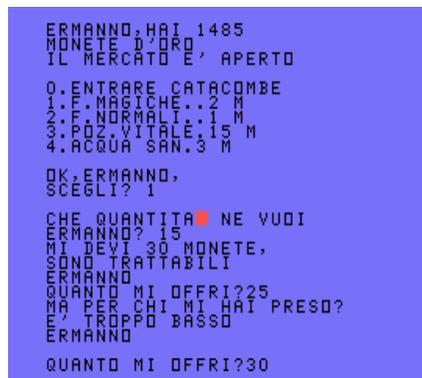
di Ermanno Betori

Negli articoli presenti sui numeri scorsi ho descritto due giochi Role Play Game per il computer MSX. Qui invece vi descriverò il mio percorso iniziatico al piacere del gioco RPG parlandovi dei giochi creati sul TI99 che mi mostrarono la luce…

Cominciamo dal descrivere la mia persona informaticamente parlando… anno domini 1983, io e il mio rapporto con il computer all'epoca si può riassumere con questa vignetta.. (p.s. io sono l'ominide a destra).

Perciò feci quello che tutti si aspettano .. cercare spiegazioni, programmi (alias giochi) possibilmente gratis… ooopsss piccolo problema spazio-tempo.. all'epoca non esisteva internet, ne vi erano le bbs, non vi era nessuna forma di comunicazione veloce… e l'unica forma di divulgazione era tramite la RIVISTA!!! SI! La santa rivista! Fonte dispensatrice del sapere informatico, sotto forma di pagine di listati di programmazione basic. Pertanto iniziai il mio apprendimento informatico partendo dal basso… molto basso… digitando come un dattilografo i programmi basic.

Dopo aver scoperto che un programma digitato, possibilmente molto lungo e funzionante, non si mantiene se si spegne il computer .. (sì .. la scoperta della memoria Ram fu cosa alquanto dolorosa e traumatica), passai allo step successivo.. l'acquisto della memoria di massa.. alias il registratore a cassette che mi diede la sicurezza che tutti i programmi digitati non sarebbero andati perduti nel tempo, come lacrime nella pioggia.

Perciò arditamente digitai il listato mammuth (quasi 5200 righe di codice) del gioco "Catacombe".

Ora all'epoca non conoscevo minimamente cosa fosse un gioco di avventura e/o RPG, pertanto dopo ore di battitura mi ritrovai un gioco che presentava una grafica minimale e poca interazione.. vedi le seguenti schermate grafiche:

Dopo aver perso quasi subito enne volte, imparai (finalmente) le basi di ogni gioco RPG o di avventura:

1 - Mai andare allo sbaraglio!

2 - I mercanti esistono, perciò compra quelle maledette frecce!!

3 - Alla Morte e Fantasma l'acqua santa bene non fà!

Ora parlando tecnicamente stiamo parlando di un gioco scritto in TI-Basic che è il linguaggio base del computer TI99/4A, molto lento, che presenta molti deficit tra cui la mancanza della gestione degli sprite. Eppure anche con tutte queste limitazioni, questo gioco rispecchia la base di ogni RPG. Vi è l'eroe, i mostri, il tesoro. Sì! All'epoca nessuna principessa da salvare.. (Mario bros bye–bye), solo vil pecunia da trovare, ed una forma minimale di iterazione con il commerciante per gli acquisti di armi ecc…

[Grosso problema dare un voto in quanto è stato il mio primo gioco in assoluto.. a livello nostalgia 100% prima giocavo con selce e pietra focaia.., obbiettivamente oggi 30-40% considerando tecnicamente su quale macchina lavora e con quale linguaggio.]

Anni dopo scoprii che il gioco "Catacombe" pubblicato su una delle più note riviste di settore dell'epoca era una traduzione e rivisitazione nemmeno corretta del gioco "Quest" pubblicato negli U.S.A. Esempio è la traduzione all'epoca della parola Ghoul con Morte. Infatti salvo pochissime persone conoscitori di demonologia (vedi preti, esorcisti, etc etc), nel nostro Italico folclore di racconti paurosi, troviamo spesso fantasmi, spettri, streghe, ma molto molto raramente se non mai, si è parlato del "ghoul", che sarebbe uno spirito demoniaco che mangia i corpi delle persone decedute. Questo piccolo esempio dimostra gli errori di traduzione che spesso soffrivano all'epoca i giochi RPG e le avventure testuali.

Gli screen shots confermano ampiamente che negli anni 80 in Italia non esisteva il concetto di copyright informatico.

Altri giochi RPG creati sulla falsa riga di "Quest", sono stati "Catacomb" e "Remzak", prodotti di discreta qualità ma totalmente sconosciuti in Italia… infatti il grande problema della mancanza di software che si aveva specialmente nel primo lustro degli anni 80, era non vi furono molti interscambi con club, gruppi o software house estere, cosa che cambierà nel decennio successivo.

Screenshot da **Catacomb**:

Screenshot da **Remzak**:

Come vediamo ambedue i programmi hanno gli elementi essenziali del gioco RPG in particolare Remzak che presenta una maggiore varietà di armi, elementi magici tesori ecc..
[Più o meno come il precedente diciamo sul 50-60% considerando che tecnicamente sono più complessi.]

Quando questi giochi esistevano negli States.. il sottoscritto doveva ancora evolversi, in quanto era fermo alla copia (piratata) del gioco "Quest"! Ma grazie all'ingegno di Italici programmatori, superai lo stallo temporale e mi addentrai nel mondo RPG acquisendo il livello "Novizio".. I programmi Ti-Basic che mi diedero lo skill superiore furono "Castello notte scura" e , "Monster Manor". Uno mi allenò su come sopravvivere in un labirinto colmo di trappole, l'altro in un mondo simulato. Infatti castello notte scura, presentava la particolarità di presentare la mappa di un mondo chiuso dove esistono città , colline , laghi etc.. che interagiva con il giocatore oltre alle solite caratteristiche tipiche dei RPG.
Esempio: sulle colline è più facile trovare giganti in grande quantità …

Finalmente stavo acquisendo esperienza nei giochi, e grazie al Extended Basic più espansione di memoria che permetteva una programmazione superiore sul TI99/4A giocai a vari RPG che mi fecero una evoluzione rapida dandomi lo skill advanced.
Ero entrato nella storia…

I giochi che mi aprirono il nuovo mondo furono "Heint", "Old Dark Caves", "Legends" e "Living Tomb".
Questi ultimi 4 programmi rappresentano le basi di come i giochi RPG si evolsero.. infatti partiamo da Heint.

DER MAGIER ARAGON WOLLTE DIE
HERRSCHAFT UEBER FRIEDLAND.
IN EINEM ELEMENTAREM KAMPF
UNTERLAG ER DEN SOEHNEN
DER DONNERGILDE. DOCH NOCH
BEVOR DER DONNERVATER ARAGON
DIE MACHT DER MAGIE NEHMEN
KONNTE VERFLUCHTE DIESER
FRIEDLAND UND VERDUNKELTE
DIE SONNE.

Programmato in X-Basic presenta le novità grafiche degli sprite e come interazione l'eroe non acquisisce in automatico i livelli, cioè aumenta le caratteristiche tipo forza, intelligenza, destrezza, ma per fare ciò deve pagare .. vi ricorda nulla?? No?? Pensateci bene .. ritroveremo questa caratteristica in molti giochi di ruolo moderni con l'eroe che paga alle gilde l'allenamento in forza, destrezza, magia, l'uso delle armi ecc..
(ricordatevi il RPG della Thalion Amberstar su Amiga / Atari ST)
[Abbiamo si tecnicamente una evoluzione ma siamo lontani dalla eccellenza diciamo sul 60-70% considerando che fù programmato da una sola persona.]

Altro punto di forza di gioco di avventura / RPG programmato in x-basic è stato Old Dark Caves che avrà pure un seguito. Qui il gioco presenta più uno stile arcade rimanendo con un background RPG. Infatti abbiamo all'inizio del gioco una scelta di cosa comprare per equipaggiare il nostro eroe. Infatti abbiamo diversi tipi di armature, di armi, di magie e di cibo. Cosa che influisce sulla capacità combattiva e reattiva del nostro eroe oltre alla nostra (scarsa) bravura nel eseguire combattimenti contro svariati mostri.

[Abbiamo tecnicamente una ulteriore evoluzione ma con un concetto diverso, come idea si avvicina a Hylide, ma rimanendo inespressa.. voto tra il 70 e 80%.]

Gli ultimi due giochi che ricordo vivamente furono Legend e Living Tomb, rappresentano idealmente la trasposizione sul TI99/4A di giochi che hanno fatto la "Storia" (con la esse maiuscola) dei Role Play game.. parlo di Ultima e Dungeon Master che sono ancora oggi considerate pietre miliari.
Che dire di Legends..

.. una volta vi era uno spot televisivo che diceva "basta la parola" qui si può parafrasare con "basta lo sguardo".

Legends (che avrà pure un seguito) è ad oggi IL RPG stile Ultima definitivo per TI99/4°. Possiede quasi tutte le caratteristiche del più blasonato Ultima, compresa una elevata difficoltà che eleva spiritualmente chi riesce a completarlo.
[Abbiamo tecnicamente il massimo di come si poteva concepire un RPG negli anni 80. I programmatori della Asgard all'epoca ci buttarono veramente il sangue per ottenere una grafica del genere usando come linguaggio non l'assembler puro ma il x-basic anche se aveva chiamate a routine assembler, infatti sul TI si definisce tale cosa come tecnica di programmazione assembly. Voto? Al momento tra il 90 e 100%. E' la gioia del giocatore RPG vecchio stile.

Finiamo la mia avventura con gli RPG sul Ti99/4° presentando Living Tomb…

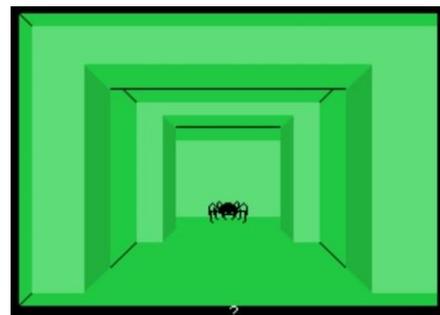
Scheda di approfondimento TI99/4A

Il Texas Instruments TI-99/4A, meglio noto come TI-99/4a, è stato uno dei primi home computer, cioè computer per uso casalingo a larga diffusione.

Prodotto dalla Texas Instruments e commercializzato a partire dal giugno del 1981, il TI-99/4A era una versione potenziata del modello TI-99/4 commercializzato alla fine del 1979. Il TI-99/4A supportava sin dall'inizio la scrittura in minuscolo (a differenza di altre macchine dell'epoca che richiedevano una specifica espansione) e una tastiera a piena escursione.

Il TI-99/4A si basa sul TMS-9900, una CPU a 16 bit con clock a 3 MHz. Costruito intorno ad un unico blocco, che racchiudeva al suo interno la CPU, la scheda madre e lo slot a cartucce (SSC Solid State Cartridges), disponeva tra i vari optional anche di un drive per floppy da 5.25", una scheda seriale RS-232 munita di due porte seriali e una parallela, una scheda P-Code per il supporto al linguaggio di programmazione Pascal, una stampante termica, un accoppiatore acustico, unità a nastro per il salvataggio e il caricamento dei dati (su normali musicassette), una coppia di joystick e un'espansione di memoria da 32 kB. Particolarità della macchina era quella di essere venduta provvista già di monitor (una versione modificata di un TV-Color Zenith da 13"), in quanto l'adattatore RF per il collegamento ad una normale TV non ottenne mai la certificazione FCC. In Italia era venduto senza monitor e l'uscita video era pienamente compatibile con i normali televisori; la grafica era basata su 16 colori e una risoluzione di 256 x 192 pixel, organizzata in 32 colonne x 24 righe, con caratteri (ASCII o definibili dall'utente) a base 8 x 8 pixel. Altra particolarità per l'epoca era la presenza di un sintetizzatore vocale opzionale, esattamente come quello presente nell'Atari 2600, che permetteva di dotare il software o i giochi di sintesi vocale. Il sintetizzatore veniva fornito gratuitamente ai clienti dopo l'acquisto di un determinato numero di cartucce, e venne ampiamente adoperato da parecchi videogiochi scritti dalla stessa TI.

Il linguaggio di programmazione predefinito era il Basic, ma era possibile estenderne le capacità mediante l'uso di una cartuccia contenente l'Extended Basic, una versione provvista di un set di istruzioni più ampio, ottimizzato e con possibilità di gestire gli Sprites.

Fonte: https://it.wikipedia.org/wiki/Texas_Instruments_TI-99/4A

Come prima descritto vuole essere un tentativo di copiare l'idea di Dungeon Master realizzando un RPG stile 3-D, ma cambiando radicalmente il modo di interazione rendendolo meno arcade dandogli un sistema di gestione più complesso ma meno intuitivo.

[Abbiamo uno dei pochi RPG stile 3D presenti per il TI99.. come voto diciamo tra il 80 e -90% considerando che anche questo programma fu creato da una sola persona.]

Concludendo, devo ringraziare i vari programmatori che crearono questi giochi che aprirono alla mia vista questi nuovi mondi, e che fecero indirettamente aumentare la mia curiosità su quale background culturale si basavano questo tipo di giochi. In pratica grazie a loro ho letto il libro il "signore degli anelli" di Tolkien.

Un saluto a tutti i giocatori che almeno una volta si sono immedesimati nell'eroe!

Oggi la scimmia non esiste più.. è stata sostituita da un vecchio avventuriero che volentieri vi aspetta alla locanda per una birra in amicizia.

Se volete approfondire la storia del **TI99/4A** vi ricordo che potete trovarla negli articoli:
Breve storia del TI-99 – parte 1 *– Numero 2*
Breve storia del TI-99 – parte 2 *– Numero 3*
su www.retromagazine.net
sempre a firma di Ermanno Betori.

Console 8bit: CBS Colecovision

di Starfox Mulder

La prima console di massa fu l'Atari VCS, il primo vero competitor fu il Mattel Intellivision e poi…arrivò il pezzo da 90. Il Colecovision si presentò nella stessa maniera con cui il Neo Geo fece dieci anni dopo: con tutta l'arroganza che la sua potenza si portava dietro.

Andiamo come sempre per gradi.

CoLeCo era l'acronimo di Connecticut Leather Company, un azienda nata nel 1932 negli States, col preciso scopo di vendere stivali e borse in pelle. Nel 1968 acquistò la Eagle Toys e da lì ci volle poco perché entrasse nel mondo dei videogames grazie al suo Coleco Telstar, un simil Pong come ce ne erano tanti all'epoca. Il vero salto di qualità lo fece nell'estate del 1982, quando introdusse la console ad 8 bit Colecovision, venduta in bundle con la miglior versione possibile di Donkey Kong (per l'epoca) e con un parco titoli comprendente altri undici giochi, a cui se ne andarono ad aggiungere dieci extra entro fine anno. L'impatto grafico rispetto alla concorrenza fu incredibile. Dopo anni di delusioni per tutti coloro che speravano di rivivere in casa le esperienze della sala giochi, trovandosi poi a convivere con pochi pixels mal assortiti, stavolta il feeling era davvero il medesimo. Poco importava che mancassero alcuni piccoli elementi ed un intero livello: Donkey Kong era lo stesso a cui potevamo giocare al bar. In un niente Colecovision divenne sinonimo di Arcade e l'azienda non sperava di meglio.

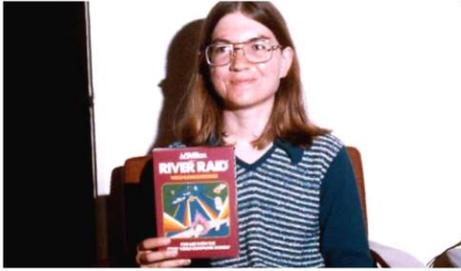
Zaxxon, Lady Bug, Spy Hunter, il parco titoli cresceva in continuazione mostrando ogni volta di più le potenzialità della macchina. Paragonarla al VCS in realtà non sarebbe etico, la vera rivale per l'epoca era diventata già il 5200 tra le macchine Atari eppure, nonostante le maggiori capacità di quest'ultima, lo scontro fu mono-direzionale. Parlavo di potenza, ed il Colecovision ne aveva da vendere. La CPU NEC D780C-1 (clone dello Zilog Z80A) ed il processore video Texas Instruments TMS9928A erano gli stessi del SG-1000 Sega (il papà del Master System) e dei computer MSX, motivo per cui diversi giochi per home pc videro la loro trasposizione console sul solo Coleco. Il controller standard assomigliava al pad Intellivision (con tanto di mascherine da inserire sul tastierino numerico) ma con l'aggiunta di un vero joystick al posto del disco direzionale. Non ci volle molto che la Coleco distribuì diversi add-on tra cui: il "Super Action Controller", uno dei pad più strani e performanti di sempre dotato di un impugnatura simil pistola e con joystick, tastierino e rotella simil trackball sulla parte superiore, il "Roller Controller", una trackball vera e propria ed infine diversi moduli di espansione atti a rendere la console compatibile con i giochi dell'Atari VCS e di altre apparecchiature dell'epoca. Si poteva fare? Secondo Atari No, infatti andarono in causa ma vinse Coleco. La vita commerciale della più potente console dell'epoca fu però brevissima dato che venne distribuita poco prima della grande crisi del mercato videoludico. A metà 1984, appena due anni dopo il lancio, la Coleco chiuse la sua divisione videogiochi, lasciando la console con oltre 140 titoli all'attivo ed una pletora di appassionati delusi ma fieri della loro rolls royce ad 8 bit.

Alla prossima console!

CARATTERISTICHE TECNICHE

Produttore	Coleco Industries, Inc.
Tipo	Home Video Game Console
Generazione	Seconda
In vendita	Agosto 1982
Dismissione	1985
Supporto	Cartuccia
Unita' vendute	Oltre 2 milioni

FONTE: HTTPS://EN.WIKIPEDIA.ORG/WIKI/COLECOVISION

RIVER RAID

Carol Shaw...

...si laureò in ingegneria elettronica a soli 22 anni e venne subito assunta da Atari dove contribuì alla creazione di celebri giochi come 3D Tic-Tac-Toe o Super Breakout.

Percorsi alternativi

Scordatevi la linearità del fiume in versione Atari. Su Colecovision le coste sono assai più frastagliate e non di rado rappresenteranno un buon motivo per morire prima del tempo.

GIUDIZIO SUL GIOCO

GIOCABILITA'

85%

Divertente ed assuefacente ma a patto di utilizzare un pad migliore di quello Coleco.

LONGEVITA'

80%

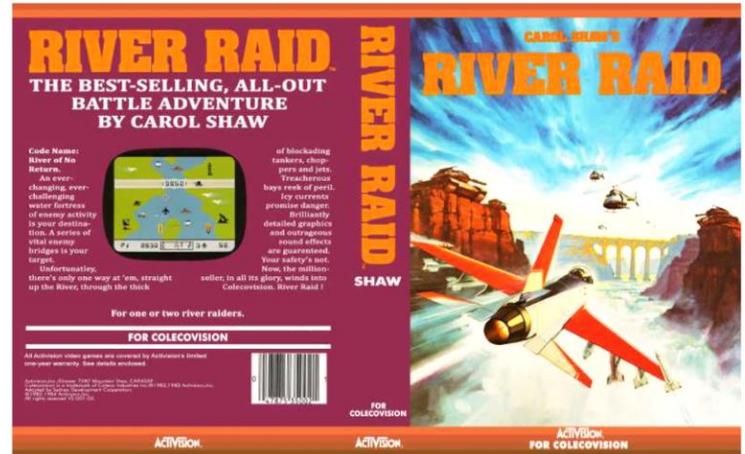
L'inseguimento del record è un ottimo incipit per ogni retrogamer che si rispetti ma alle lunghe.

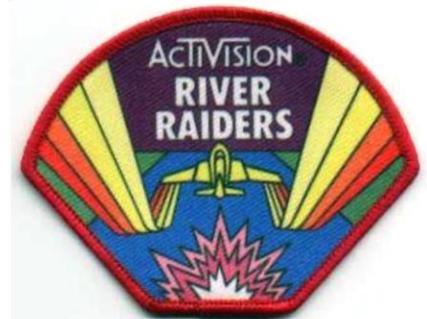
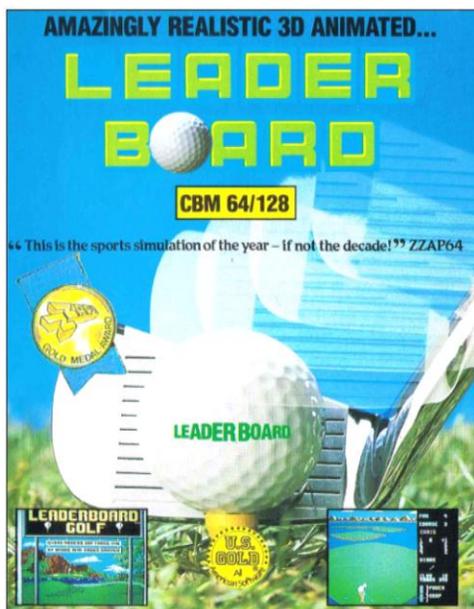
River Raid

Sydney Development Corp. - Anno 1984 - Piattaforma CBS Colecovision

E' da poco passata la Festa delle donne quindi trovo opportuno tributare la prima grande programmatrice videoludica della storia parlando del gioco che preferisco tra quelli partoriti dal suo estro. Carol Shaw si laureò in ingegneria elettronica a soli 22 anni e venne subito assunta da Atari dove contribuì alla creazione di celebri giochi come 3D Tic-Tac-Toe o Super Breakout. La sua permanenza in azienda non fu delle più brevi ma anche lei nel 82 si era stancata della politica sempre più restrittiva del signor Kassar e decise di cambiare aria. Alla Activision, auto-elettisi paladini delle terze parti, non si fecero scappare l'occasione e l'acciuffarono subito dopo la mancata assunzione presso Imagic, dove l'avevano etichettata "inadatta ai giochi di azione". Carol pensò bene di fargliela vedere quanto era inadatta e sfornò uno dei migliori e più geniali sparatutto di seconda generazione mai creati. Originariamente pubblicato su Atari 2600, River Raid vide nella trasposizione per Colecovision (a cura di Sidney Development Corp.) un implementazione non solo tecnica ma anche di gameplay, vista la maggior varietà del percorso da seguire.

Il giocatore viene chiamato a comandare un aereo diretto lungo le rive del non ritorno, un terribile fiume costellato di insidie quali: navi nemiche, elicotteri, aerei, ponti da abbattere e chi più ne ha più ne metta. Il nostro jet si muove in autonomia ma potremo rallentarne o accelerarne il percorso premendo in su o in giù sul joystick, così come muoverci sulla linea dell'orizzonte tramite le direzioni destra o sinistra. I tasti laterali del controller serviranno all'ultima utile funzione, ossia quella di sparare ai nemici ed agli ostacoli che ci troveremo dinnanzi. Per fortuna il colecovision contempla il cambio di controller ed in questo caso potrebbe davvero essere utile inserire un pad a 9 pin più performante, ma la libertà è ovviamente in mano al giocatore ed al suo spirito emulativo.

Ogni tot chilometri percorsi un ponte ci sbarrerà la strada ed abbattendolo piazzeremo una bandierina virtuale sul nuovo checkpoint da cui ripartire in caso di prematura morte per collisione o incidente simile. Le vite a nostra disposizione saranno inizialmente quattro ma aumenteranno ogni 10000 punti. "Easy to learn, Hard to master" ovviamente, e mentre le partite si susseguiranno ci malediremo per non aver partecipato al contest Activision dell'epoca, quello cioè per con cui saremmo stati ricompensati da una splendida toppa nel caso ci fossimo fatti un autoscatto (non si chiamavano ancora selfie) assieme al record raggiunto di almeno 15000 punti.

Lungo il percorso non dovremo mai scordarci di fare carburante passando sopra le taniche di benzina per il rifornimento in volo, pena un atterraggio prematuro assai doloroso, ed ecco che il nostro personale record cesserà di crescere solo per sopraggiunta noia, cosa che di certo non accadrà tanto presto!

di Starfox Mulder

LEADER BOARD

Inizio del gioco

La prima cosa che ci viene richiesta in Leader Board e' il numero dei giocatori.

GIUDIZIO SUL GIOCO

GIOCABILITA'

95%

Il sistema di controllo e' semplice ed immediato. Cambiare mazza e' facile come avere un caddy sempre a portata di mano! Ma nonostante l'apparente semplicita' la profondita' del gioco e' immensa.

LONGEVITA'

90%

Sono passati 32 anni, la grafica delle simulazioni sportive ha fatto passi avanti notevoli, ma nonostante l'aspetto grafico la giocabilita' di Leader Board non ne ha risentito minimamente. Se non lo avete mai giocato, dovete provarlo almeno una volta nella vita; se invece eravate appassionati giocatori negli anni '80, ricarite il gioco e vi sorprenderete a divertirvi giocandolo nuovamente!

Leader Board

Access Software - Anno 1986 - Piattaforma Commodore 64

E' il 1986 e Access Software se ne esce con un gioco del Golf destinato a riscrivere la storia di questo sport su tutti gli home computer. Come recita il manuale allegato al gioco, Leader Board è il più realistico gioco del Golf mai sviluppato per home computer perchè vi dà la prospettiva ed il punto di vista reale del giocatore ad ogni colpo! Una volta tanto ci troviamo d'accordo! Non si tratta di una pura citazione commerciale, Leader Board fa tutto quello che promette ed anche di piu'! Non e' solo un gioco, ma prima di tutto una vera e propria simulazione dello sport del Golf!

Lo scopo del gioco del Golf è piuttosto ovvio, ma per chi vivendo sulla Luna (...con Planet Golf non è piu' vero nemmeno questo! NdR) non sapesse in cosa consista una partita di Golf, ve ne parlerò brevemente. Lo scopo del gioco è quello di infilare la pallina in buca con il minor numero di colpi possibile, evitando accuratamente gli ostacoli naturali ed artificiali presenti nel percorso, calcolando con cura forza, direzione ed effetto da imprimere alla pallina ad ogni colpo.

Cominciare a giocare e' piuttosto immediato, tanto che la curva di apprendimento e' estremamente veloce. Ma, le variabili in gioco sono talmente tante che, soprattutto nel livello più difficile, il Professional, il tutto risultera' tutt'altro che semplice aumentando considerevolmente la profondità del gioco stesso.

Ma andiamo con ordine. Leader Board ci accoglie chiedendoci il numero di giocatori, da 1 a 4, ed in base a quello il/i nome/i di ognuno di essi. Purtroppo il nome e' limitato ad 8 caratteri, molto comune sul C64. Subito dopo siamo invitati a scegliere il livello di difficolta' (vedere tabella sottostante).

Novice	Vento ininfluente. Punto di battuta sempre centrale.
Amateur	Traiettoria influenzata dall'effetto. No vento.
Professional	Vento e punto di battuta influsicono sulla traiettoria della pallina.

Dopodiche' procediamo a scegliere il numero di buche sulle quale vogliamo giocare. Da un minimo di 18 buche ad un massimo di 72, passando per i valori intermedi di 36 e 54. Ma non e' ancora finita, dobbiamo scegliere quale percorso giocare. Anche qui ce ne sono ben 4, ognuno con delle caratteristiche e peculiarita' ben distinte.

Finalmente siamo pronti a giocare. Il C64 disegnera' per noi percorso della prima buca e ci posizionera' immediatamente alle spalle del nostro alter ego elettronico, pronti a colpire la pallina.

Come ormai in ogni gioco del Golf che si rispetti, ma con Leader Board come apripista, ad ogni colpo dovete scegliere quale mazza utilizzare prima di colpire la pallina. Ed attenzione a scegliere la mazza giusta, perche' come potete vedere dalla tabella allegata, ognuna di esse ha una portata ben precisa, che unita alla forza ed alla precisione con cui viene colpita la pallina, imprimera' alla stessa una traiettoria ben precisa.

Tabella della portata delle mazze (in yard):

Mazza	Minima	Massima
1W	156	271
3W	135	245
5W	128	234
1I	110	220
2I	100	210
3I	88	202
4I	70	189
5I	67	181
6I	55	169
7I	50	153
8I	36	138
9I	26	117
PW	11	83
Putter	1 Inch	64 Feet

Dopo aver scelto la mazza passiamo finalmente a colpire la pallina e, come si vede dalla tabella della difficolta', il livello che abbiamo scelto influenzera' in modo determinante il nostro comportamento durante il colpo.

Sul green

Dobbiamo dosare la forza del putter per infilare la palla in buca o almeno avvicinarci .

Attenzione al bosco centrale

Per avvicinarci alla buca dobbiamo aggirare l'ostacolo centrale rappresentato dal bosco di alberi.

Il gioco si fa duro...

Per raggiungere la buca dovremo evitare il laghetto ed il bunker di sabbia.

La pendenza del green

Vedete l'asticella che proietta l'ombra? Serve ad indicare la pendenza del green e quindi a calcolare la forza e la direzione da imprimere con il putter per completare la buca.

Con il livello Novice ci bastera' calcolare ad occhio la potenza da destinare al colpo, mentre per i livelli Amateur e Professional il discorso e' un po' piu' complesso.

Figura. 1 - Indicatore della potenza e dell'effetto. Zzap nr. 3 - Luglio 1986

Come si vede infatti in Figura 1, premendo il pulsante di tiro sceglieremo prima la potenza, rilasciando la pressione quando l'indicatore della forza e' al livello desiderato, e subito dopo dovremo premere nuovamente il pulsante per scegliere l'effetto da dare alla palla. Difficile? No, basta prenderci la mano e tutto verra' naturale.

Ogni buca ha un PAR, cioe' un numero di tiri necessario per imbucare la pallla. Ovviamente se siamo estremamente bravi possiamo completare la buca anche con un numero minore di tiri e migliorare di conseguenza il nostro punteggio.

Nome	Significato
Bogey	Buca in par +1 colpo
Par	Anche detto Even
Birdie	Buca in par -1 colpo
Eagle	Buca in par -2 colpi
Albatross	Buca in par -3 colpi
Condor	Buca in par -4 colpi
Phoenix	Buca in par -5 colpi
Hole in	Buca in un colpo solo

Passiamo qiundi a parlare del reparto grafico e sonoro di Leader Board.

Il sonoro e' veramente basilare, ma questo non significa che non sia ben fatto. Nessuna musichetta di sottofondo a distrarre il giocatore, proprio come in una partita di Golf reale, la concentrazione richiede silenzio. Ma il rumore della pallina che si infila in buca provoca sempre un certo piacere!

Quello che invece colpisce in Leader Board e' la grafica! La grafica vettoriale 3d ricostruisce il campo di Golf in maniera precisa e sufficientemente dettagliata come mai non era stato fatto ancora per una simulazione golfistica.

I 72 percorsi sono vari e disseminati di ostacoli naturali ed artificiali ed il nostro Commodore 64 li disegna con una facilita' ed una velocita' sorprendenti! E' veramente affascinante vedere come ogni percorso viene tracciato aggiungendo particolari su particolari. Che dire tanto di cappello ai programmatori della Access Software.

Conclusioni.

Leader Board e' un classico! Ha riscritto le regole delle simulazioni golfistiche alzando l'asticella ad un punto tale che per molti anni ha segnato il termine di paragone per questo sport. Sono passati 32 anni e ne abbiamo viste di tutti i colori, ma il gioco rimane comunque godibile. Certo la grafica non fa piu' balzare sulla sedia, ma nonostante tutti i limiti tecnici non ci si fa assolutamente caso. La giocabilita' invece rimane su livelli eccezionali. Che dire, provatelo se non lo avete mai giocato, rigiocatelo se lo conoscete gia', sara' una bella riscoperta!

di Francesco Fiorentini

Dov'e' la buca?

R-TYPE

R-Type

Images Design / Electric Dreams - Anno 1988 - Piattaforma Atari ST

La schermata di caricamento

Promette bene, no?.

Un altro giro di giostra

Uno dei tanti momenti iconici del titolo Irem

Dbkeratops

Il Mostrone di fine livello™ per antonomasia

Creature Cave

Certo che ne avevano di fantasia i grafici Irem!

La minaccia arriva dal futuro. La razza dei Bydo, derivata da esperimenti genetici condotti dai terrestri nel 26° secolo, giunge nel nostro tempo tramite un varco spazio-temporale mettendo in gravissimo pericolo la sopravvivenza del genere umano. L'ultima speranza è riposta nell'avanzatissimo caccia R9, inviato a portare un disperato attacco suicida contro il potentissimo impero Bydo. Le devastanti armi della nostra navicella dovranno affrontare enormi e mostruose creature biomeccaniche, una gigantesca astronave da combattimento, un gran numero di navicelle, mechs, walkers lanciamissili, postazioni di difesa, droidi e letali creature aliene di ogni forma e dimensione, sparse in 8 terribili locations irte di insidie. L'estrema risorsa del genere umano è dunque posta nelle vostre mani: BLAST OFF AND STRIKE THE EVIL BYDO EMPIRE!

R-Type è uno shoot 'em up a scorrimento orizzontale e ambientazione spaziale realizzato da Irem nel 1987.

Le ambientazioni proposte inseriscono nella tradizionale iconografia fantascientifica venature horror mutuate dal film Alien e ispirate al design gigeriano, giunto alla sua esaltazione e massificazione proprio nel capolavoro di Ridley Scott. Il gioco unisce, dunque, le suggestioni di serie come Gundam, di cui riprende il mech design e il look delle esplosioni, e i suddetti echi gigeriani in un azzeccato cocktail che, complice l'eccezionalità della realizzazione tecnica, colpisce la fantasia dei videogiocatori (sulle custodie delle versioni per home computers campeggiava It's Mechanical, It's Biological... It's behind YOU, a fare, in un certo senso, il pari con il sottotitolo italiano di Alien Nello spazio nessuno potrà sentirti urlare).

R-Type è un grandissimo e meritato successo. Un titolo geniale che "azzera" il genere degli shoot 'em up e, sostanzialmente, ne "resetta" le caratteristiche tramite l'introduzione di tutta una serie di novità più o meno clamorose, tanto di ordine concettuale e inerente al gameplay (si sarebbe potuto dire che, come i Bydo, R-Type veniva dal futuro), quanto di attinenza strettamente tecnica e relativa alle avanzatissime soluzioni grafiche costituenti un look che, nel 1987, non può non lasciare profondamente impressionati. Il coin-op Irem "detta legge" per parecchi anni dopo la sua realizzazione. Si potrebbe affermare che il titolo Irem funge da "supermercato delle idee" per moltissimi sparatutto, i cui autori si "ispirano" a questa pietra miliare per svilipparne determinati spunti o aspetti, con i benefici che derivano dalla costante evoluzione dell'hardware. Sul fronte delle idee, dunque, R-Type compie un'audacissima esplorazione riguardo alle meccaniche di gioco degli shoot 'em up. L'indubbia creatività degli autori fa sì che questo sparatutto sia considerato dai videogiocatori e dai programmatori di futuri titoli analoghi come un imprescindibile punto di partenza, in cui è già delineata la maggioranza delle tipologie di azione sviluppabili in un titolo del genere.

Vediamo dunque le numerose novità introdotte da questo arcade hit. Il sistema di armamenti include un modulo di difesa, il Force Pod, che protegge la navicella dai colpi standard e può essere lanciato verso il nemico, recuperato ed, eventualmente, posto sul retro dell'R9. Tutti i potenziamenti raccolti, con l'eccezione di due mini-pods fissi e relativi laser posizionati sui lati della navicella, sono legati al Pod e consentono all'R9 di fare fuoco in avanti, se l'unità è in posizione frontale, come dietro, se essa è collocata sul retro del veicolo. In diverse situazioni le manovre di lancio, sgancio e aggancio del Force Pod sono assolutamente fondamentali e devono essere eseguite con velocità e tempismo. La navicella, poi, è dotata di uno sparo base che può "caricare", tenendo premuto il pulsante di fuoco, e rilasciare in un'onda d'energia distruttiva detta Beam. La potenza di tale arma è direttamente proporzionale alla quantità di "carica" accumulata, visualizzata dalla barra di caricamento posta nella parte bassa dello schermo. Tra i potenziamenti, oltre ad un devastante e molto scenografico sparo a "doppia onda", spiccano i laser "a rimbalzo" che consentono di colpire i nemici utilizzando gli elementi del fondale e il fascio energetico che si genera dai lati del Pod e prosegue lungo i contorni delle superfici. Le armi sopra descritte introducono ulteriori stimolanti novità che aumentano quelli che, al limite, si potrebbero chiamare "elementi tattici" e si basano sulla marcata e accuratamente implementata interattività degli scenari del gioco.

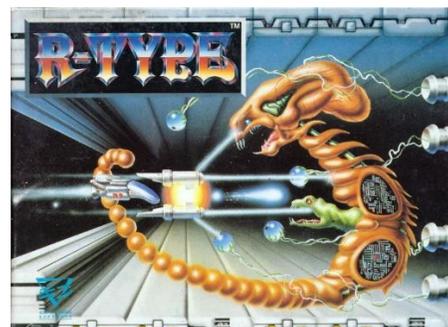
R-Type, poi, sancisce e "istituzionalizza" la figura del boss finale (quello che gli "addetti ai lavori" delle sale giochi chiamavano: "Il Mostrone di fine livello™") e ne "offre" ai laser dei videogiocatori un campionario che rimarrà a lungo insuperato per suggestione visiva, caratterizzazione e livello di sfida: l'indimenticabile "Alien" del 1° stage, con la sua coda multi-sprite (realizzata con l'inedita tecnica di animazione modulare che abbina numerosi elementi grafici e ne sincronizza il movimento), l'impressionate "cuore alieno" del 2°, con relativo lunghissimo verme, l'imitatissima astronave da combattimento del 3° (l'unico nemico da abbattere pezzo per pezzo del terzo stage che, quindi, non ha un vero e proprio boss finale), l'agguerrito modulo da combattimento "tripartito" del 4°, l'incubo biomeccanico del 5°, la claustrofobica "giostra" spaziale del 6°, la pericolosissima cascata di rifiuti spaziali con relativo "guardiano" del 7° e il super protetto

alieno, rinserrato nella sua tana vivente, che conclude l'8°.
Gli elementi innovativi di R-Type non finiscono, però, nella creatività degli armamenti (in particolare del Force Pod e del Beam), nella suggestione dei boss finali e nell'azzeccatissima concezione e realizzazione del celeberrimo 3° stage che diventerà una vera e propria tappa obbligata di un gran numero di "eredi" del capolavoro Irem. Fondamentale per il successo del gioco è la cura nell'implementazione degli attacchi nemici, l'ingegnosità dei loro movimenti e l'accortezza (leggi: "perfidia") della loro disposizione. Le dinamiche variano peraltro moltissimo da livello a livello, con sezioni di puro blastaggio ad alto tasso di frenesia, alternate a zone dove si privilegia la precisione della manovra e, ancora, a situazioni dove è essenziale tempismo, attenzione e un pizzico d'inventiva.R-Type, dunque, è un flusso continuo di sorprese e colpi di scena in una diversificazione costante, che stupisce ancora di più se si considera lo straordinario livello della realizzazione tecnica sviluppato nell'ambito di una ROM di 8 Mbit (1 MB) con notevole ricchezza di particolari, impressionante attenzione ai minimi dettagli, ricco assortimento di nemici e gradevoli animazioni.
Questa ricchezza grafica è permessa, oltre che dall'indiscutibile perizia dei programmatori, da un hardware di tutto rispetto. Il titolo Irem, infatti, gira su una scheda M72 basata sulla potente CPU Nec V30 a 8 Mhz. La M72 permette una risoluzione di 384 X 256 (nel 1987, dunque, R-Type era un titolo "HD" che permetteva un dettaglio molto maggiore dei contemporanei titoli Konami, realizzati nella più modesta 256 X 224), una palette di 512 colori con fino a 128 tonalità su schermo e lo scrolling hardware a due piani distinti che diventerà richiestissimo con la nota definizione "scrolling parallattico". I fiori all'occhiello di ordine tecnico del titolo Irem sono, dunque, l'"alta risoluzione", le notevoli dimensioni dei boss finali, le animazioni modulari (protagoniste indiscusse del 2° e 5° stage) che daranno inizio ad una invasione di tentacoli, vermi e serpenti costituiti da sprite in un gran numero di titoli successivi, i fondali in fluidissima e dettagliata parallasse (stages 2, 5, 6 e 7), l'impressionante velocità e frenesia dell'azione, l'alto grado di "affollamento" dello schemo con rarissimi rallentamenti (si limitano al confronto finale con la nave gigante del 3° stage e ad alcuni punti "caldi" del 7°), la peculiarità e l'eleganza del design, la maniacale cura posta dai grafici per la realizzazione del più piccolo particolare e la capillare distribuzione di numerosi "tocchi di classe".
Le musiche e gli effetti sonori del titolo Irem sono all'altezza dell'impeccabile comparto grafico. Il chip audio Yamaha YM2151 consente allo shoot 'em up una buona colonna sonora in sintesi FM (Frequency Modulation) e degli ottimi FX dal feeling molto "tecnologico" (particolarmente azzeccati i rumori del caricamento del beam, del frangersi dello stesso e le esplosioni "tintinnanti", da flipper, dei nemici più piccoli). Le musiche variano da particolarmente incalzanti e adrenaliniche (stages 1-4-6) a più cupe e oppressive (stages 2,3,7). Particolarmente azzeccati per concezione composizione i brani che sottolineano lo scontro con i boss ("Indian Battle") e i fulminati "attacchi" della musica suddetta e di quella del 1° stage.

La conversione di questo popolare coin-op per il 16 bit Atari è sviluppata, come quella Amstrad, a partire dal codice del porting per Spectrum.
I grafici hanno utilizzato a dovere la palette di 512 colori dell'ST e dosato accuratamente i 16 colori che il computer può visualizzare su schermo. Il risultato è un'estetica che sembra molto ben realizzata... finche non se ne "apprezza" il movimento.
Gli sviluppatori, l'Images Design, non hanno limitato l'area di gioco, rinunciando alla barra di status, che, considerate le difficoltà dell'ST nella gestione dello scrolling, sarebbe stata quanto mai opportuna per alleggerire il carico di lavoro sulla CPU.
I programmatori, inoltre, hanno incautamente mantenuto le dimensioni degli sprite molto vicine a quelle degli originali e quindi, con un'area di gioco più piccola del coin-op per i bordi che, comunque, delimitano le modalità grafiche dell'ST, hanno creato i presupposti per frequenti situazioni di "affollamento" claustrofobico.

La conseguenza di queste scelte è una cronica mancanza di fluidità dei movimenti che fa rimpiangere titoli ST contemporanei meglio concepiti come *Armalyte* e *Menace*. Lo scrolling, infatti, "arranca" con un fastidioso movimento "gelatinoso" e gli sprites sfarfallano in modo tanto marcato da compromettere gravemente la giocabilità. La fruibilità del gioco è, poi, ulteriormente ostacolata dal movimento estremamente scattoso dei colpi nemici, difficili, così, da evitare e, soprattutto, da alcuni fastidiosi bug che rendono invisibili (e quindi "letali") certi piccoli sprite in alcune situazioni di gioco.

Le musiche sono state sì riprodotte con una certa accuratezza, ma sono comunque, fruibili in alternativa agli effetti sonori (molto simili a quelli per Amstrad). I brani sono realizzati con il primitivo chip PSG (Programmable Sound Generator) a 3 canali dell'ST che, nonostante i grossi limiti, riesce a generare una colonna sonora quasi accettabile e, in alcune BGM, meno spiacevole delle corrispondenti versioni "base" MSX e Master System, dotati, in mancanza di FM units, di chip audio simili a quello dei "Sixteen/Thirty-two".

di Alessio Bianchi

GIUDIZIO SUL GIOCO

GIOCABILITA'

65%

La notevole difficoltà del coin-op è sì mantenuta anche nella conversione per Atari ST, ma per motivi legati principalmente ai problemi di fluidità, alla mediocre gestione dei proiettili nemici e delle relative collisioni e a pochi ma vistosi bug .

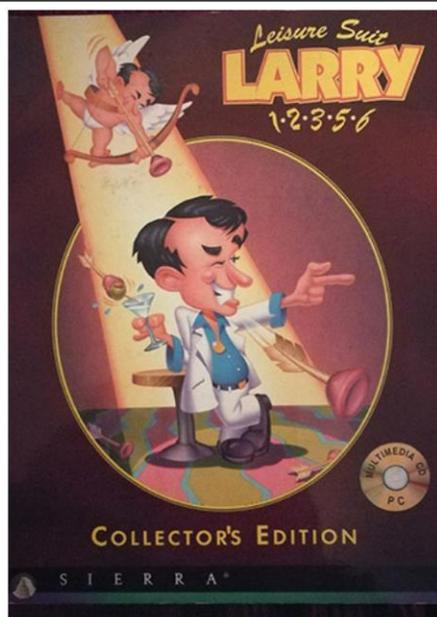
LONGEVITA'

70%

Nonostante i notevoli limiti tecnici dovuti sia a un coding tutt'altro che ottimale sia ai limiti intrinseci dell'hardware, R-Type può essere portato a termine... ma ci vuole un fisico bestiale!

Leisure Suit Larry in the Land of the Lounge Lizards

Sierra On-line - Anno 1987 - Piattaforma IBM PC/MS-DOS

LEISURE SUIT LARRY 1

La copertina da collezione…

…non avendo più la copia del primo capitolo vi propongo l'immagine dell'edizione da collezione dei capitoli 1-6 (il 4° non è mai stato pubblicato) che ho ancora.

Il Lefty's Bar

Questo è il punto di partenza dell'avventura

GIUDIZIO SUL GIOCO

GIOCABILITA'

80%

Un'avventura piacevole e divertente. Con battute a volte sboccate e un po' sessiste e morti "casuali", ma è sempre divertente perderci un po' di tempo.

LONGEVITA'

80%

Il primo capitolo di una saga di adventures grafiche a sfondo ero-comico. Con le edizioni "replica" e i capitoli successivi sono 10 giochi.

Ammettiamolo, tutti noi, nerd e teenagers degli anni '80 e '90 dello scorso secolo siamo sempre stati un po' "attratti" dal materiale "pruriginoso" che iniziava a circolare su PC e affini…dai, non siate timidi! Alzi la mano chi non conosce la "saga" di Larry Lafner…lo sapevo, non vedo nessuna mano alzata! ;)

Leisure Suit Larry è una serie di giochi a sfondo "sexy", ma più che altro comico, principalmente adventures grafiche, ideata e realizzata da Al Lowe. Inizialmente progettata per essere una trilogia, si è evoluta fino a comprendere 10 titoli e 4 programmi collegati.

Questo primo capitolo della saga è una classica avventura grafica dell'epoca: un'interfaccia grafica, ma con comandi testuali che dovevano essere immessi tramite tastiera. Ad esempio, il primo comando da usare in ogni nuovo ambiente è il classicissimo "LOOK", per prendere un oggetto "GET OBJECT", "USE OBJECT" per usarlo e così via. La grafica era piuttosto semplice, anche per l'epoca, in risoluzione EGA a 16 colori (anche se nel 1991 è stata realizzata una riedizione con grafica VGA) con una pixelatura importante.

Il protagonista del gioco, Larry, è un ometto di mezza età, bassino, cicciottello, arrogante e un po' stupido…diciamocelo: uno sfigatello che si crede un latin-lover pazzesco! Lo scopo del gioco è aiutare Larry a perdere la verginità…ehm no, volevo scrivere: a trovare l'amore.

L'avventura è ambientata nell'immaginaria città di Lost Wages, si comincia all'entrata del famigerato Lefty's bar. Subito, nelle prime battute del gioco, Larry ha la possibilità di fare sesso con una prostituta; ovviamente non perde l'occasione, esce dal bar dopo aver consumato e…bam! Viene arrestato dalla polizia per atti osceni in luogo pubblico perché ti sei dimenticato di tirare su la zip dei pantaloni….*game over!*

Ah-ah…hai capito tutto! Tutto carico e tronfio per aver scovato l'errore, ricominci l'avventura, consumi con la prostituta, ti tiri su la zip dei pantaloni e esci… e poco dopo muori a causa di una malattia infettiva…*game over!*

Again! La malattia non viene mai espressamente citata nel videogioco, ma l'alone rosa fosforescente che ti avvolge poco prima di morire ricorda una "pubblicità progresso" in voga in quegli anni.

Tranquilli, non sarà la vostra "ultima morte", infatti nel gioco, oltre a quelle "funzionali" che imparete presto ad evitare, entrando nella filosofia del gioco, ci sono delle morti casuali o quanto meno imprevedibili…tutto questo è voluto per prendere in giro le adventures grafiche prodotte in quegli anni dalla Sierra nelle quali si poteva morire per le cose più stupide. Ma tranquilli, in questo caso non vedrete il famoso e noioso "game over", ma Larry verrà preso, sezionato e triturato da un simpatico manipolo di scienziati.

Eve l'obiettivo finale dell'avventura di Larry

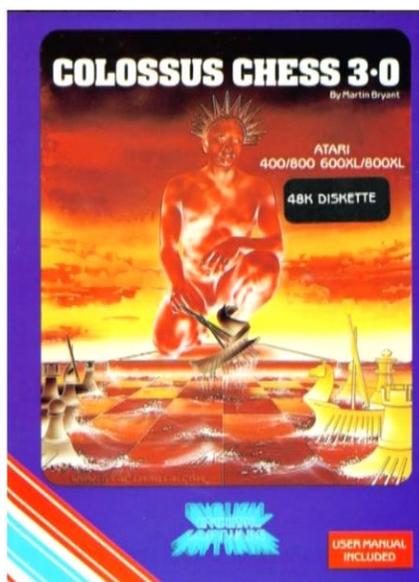
Proseguendo nel gioco, dovrete esplorare la città e per farlo dovrete usare molto il taxi e quindi spendere soldi (necessari anche per comprare vari oggetti) che dovrete procurarvi visitando spesso il casinò. Dopo mille peripezie, battute, situazioni più o meno piccanti arriverete al traguardo finale, cioè Eve, conquistatela e…godetevi i fuochi d'artificio finali!

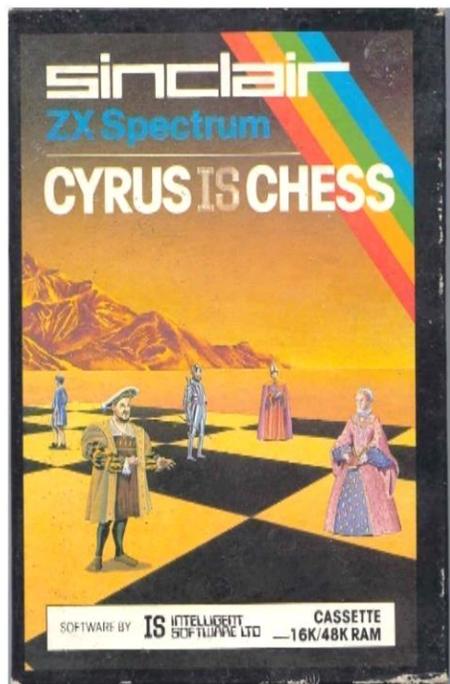
Una delle caratteristiche principali del gioco è che è una delle prime avventure che vengono raccontate da un narratore esterno che si rivolge al giocatore col "tu" come se il giocatore fosse veramente Larry. Il gioco, seppure le battute siano spesso volgari e sessiste (tipicamente anni '80) non degenera mai troppo, rimanendo una "commedia" divertente e piacevole. Non esistono scene di nudo esplicito e le scene di "sesso" sono sempre coperte da un riquadro nero "CENSORED".

di Gianluca J.R. Romani

SFIDA: CYRUS (Sinclair ZX Spectrum) VS COLOSSUS (Atari 800XL)

COLOSSUS CHESS – ENGLISH SOFTWARE UK - 1984

COLOSSUS CHESS 3.0 PER ATARI 800XL

CYRUSS CHESS – INTELLIGENT SOFTWARE - 1983

CYRUSS IS CHESS 3.0 per Sinclair ZX Spectum

Uomo VS Computer e Computer VS Computer

Cyrus Chess VS Colossus Chess, ovvero Richard Lang contro Martin Bryant. Chi sono costoro? Due programmatori britannici che alla fine degli anni Settanta e inizio degli Ottanta si erano specializzati nella programmazione di software in grado di giocare a scacchi con una forza sufficiente a mettere in crisi anche buoni giocatori classificati. La storia di macchine logiche ed elettromeccaniche dedicate al 'nobil giuoco' non è certo recente, anzi risale agli inizi del Ventesimo secolo. Nel tempo se ne sono occupati anche nomi illustri dell'informatica come Von Neumann, Turing, Wiener, Zuse, Shannon, e da sempre la sfida di battere l'uomo ha rappresentato un ottimo terreno di applicazione per matematici ed informatici (cfr. la famosa scommessa di David Levy, che nel 1968 scommise che nessun computer scacchistico l'avrebbe sconfitto entro il 1977).

Ad oggi, la "partita" fra uomo e macchina è sostanzialmente chiusa. Le macchine, o per meglio dire, i motori software scacchistici hanno avuto la meglio sui migliori giocatori umani in circolazione da almeno una quindicina d'anni. Gli *engine* sviluppati nella forma classica hanno ormai raggiunto il loro apice e programmi come Stockfish, Houdini e Komodo superano i più quotati campioni umani di circa 600 punti ELO. Sulla carta è praticamente un abisso. L'attuale campione del mondo "umano" è il norvegese Carlsen, seguito da vicino dal "nostro" Caruana (fino a due anni giocava per la Federazione Italiana, poi è passato agli USA), dagli "statunitensi" Nakamura e So, dai russi Mamedyarov, Karjakin, Kramnik e Svidler, dall'indiano Anand, ecc. e tutti da qualche anno utilizzano i motori per allenarsi ma anche per "apprendere" il loro stile di gioco.

Ma da qualche mese il settore dei software scacchistici (e più in generale dei giochi di strategia) sta vivendo una vera e propria rivoluzione copernicana: l'algoritmo basato sull'Intelligenza Artificiale denominato AlphaZero, sviluppato da DeepMind, una società AI controllata da Google, non solo è stato in grado di battere ripetutamente il campione mondiale di Go, ma ha superato brillantemente il migliore motore scacchistico in una sfida su 100 partite (+28 =72 -0, cioè 28 vinte, 72 patte e 0 sconfitte) tenutasi lo scorso dicembre 2017. AlphaZero, in sole 4 ore di allenamento contro se stesso, ha "imparato" a giocare ed ha poi stracciato Stockfish (versione 8) con 25 vittorie e 25 patte con il Bianco e 3 vittorie e 47 patte con il Nero. Un risultato inaspettato e a dir poco sensazionale per gli addetti ai lavori, che apre frontiere tutte nuove nel mondo dell'AI applicata a problemi assimilabili a giochi complessi come gli scacchi

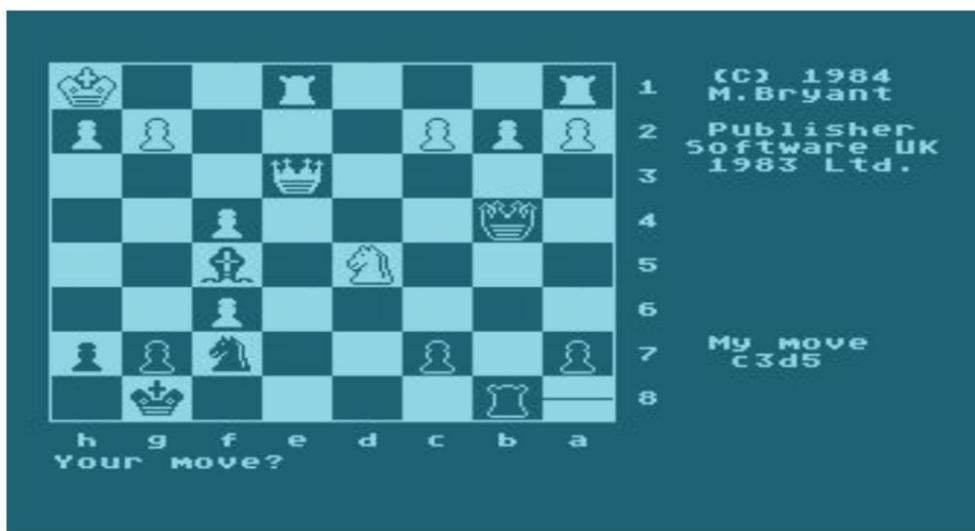
Scacchi e Home Computer: i protagonisti

Ma veniamo ai nostri amati retrocomputer. All'inizio degli anni Ottanta, con l'avvento degli home computer, programmatori come

Lang e Bryant, furono assunti dal nascente mercato del software commerciale per sviluppare i loro algoritmi di successo sulle macchine a 8-bit. I risultati possiamo apprezzarli nei due titoli che in queste pagine sono stati scelti per un match/recensione. Sì,

perché abbiamo condotto una piccola ricerca per stabilire quale dei due algoritmi fosse il più forte di allora, mettendo a confronto due delle loro migliori implementazioni: Cyrus Chess su Sinclair ZX Spectrum e Colossus Chess 3.0 su Atari 800XL.

Richard Lang cominciò la sua carriera di programmatore specializzato nel gennaio del 1981 dopo aver studiato a fondo un libro su Sargon di Dan e Kathe Sprackler che forniva un listato in Assembly Z80. Lang si mise al lavoro e trovò diversi modi per migliorare lo schema teorico su cui era basato Sargon, non solo per ottenere maggiore rapidità ma anche per implementare tecniche più avanzate ed un sistema più efficiente per valutare le posizioni raggiunte durante le partite. Il suo primo programma, Cyrus, vinse il Secondo Campionato Europero di Scacchi per Microcomputer a Londra nel settembre del 1981, con 5 vittore su 5 incontri disputati. A Lang fu subito offerto un contratto di lavoro dalla Intelligent Software (fondata da David Levy) e Cyrus IS Chess per ZX Spectrum fu il suo primo titolo commerciale.

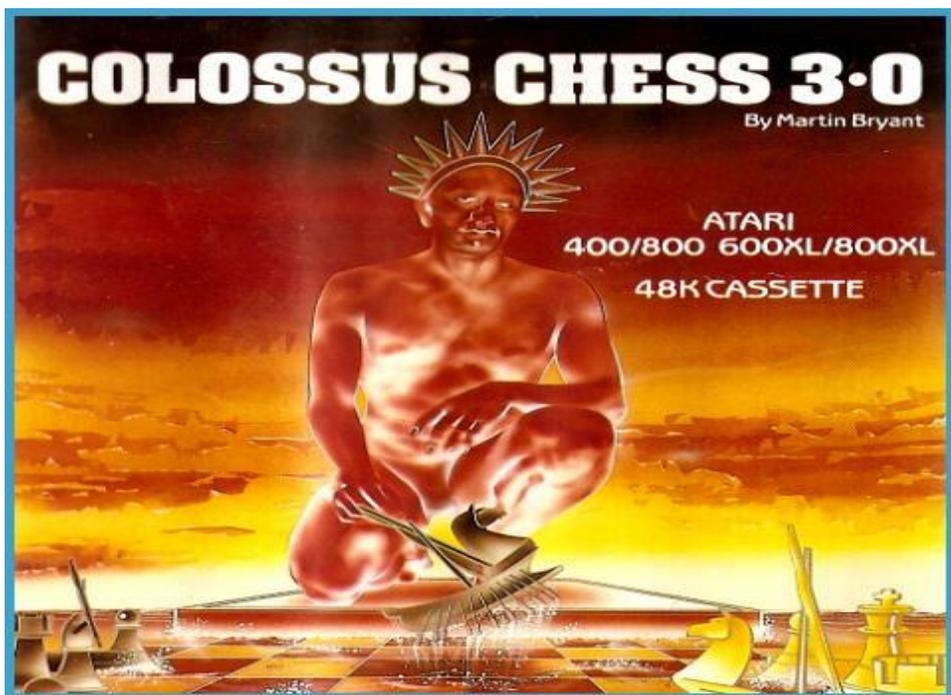
In seguito Lang si dedicò al porting di Cyrus per varie piattaforme per conto della IS per poi passare nel 1983 al nuovo programma Psion per i processori 68000 a 16 bit, la cui prima pubblicazione fu la versione per Sinclair QL. Nel 1985 Lang collaborò al porting di Psion verso la famosa serie di computer scacchistici Mephisto che, insieme alla sua emanazione software chiamata Chess Genius, dominò la scena dei microcomputer dedicati dalla metà degli anni 80 fino all'inizio degli anni 90, vincendo più volte il titolo di Campione del Mondo di categoria. Nel 1994 Chess Genius vinse addirittura una partita

rapida (25 minuti di tempo per parte) contro l'allora campione del mondo Garry Kasparov. Dal 2002 Lang dirige la sua piccola azienda che commercializza Chess Genius per varie piattaforme, in particolare per PDA e smartphone, Android incluso, ed ancora oggi molte delle idee e delle tecniche originali inserite in Cyrus vivono nelle *release* attuali.

Martin Bryant iniziò a sperimentare il suo primo programma di scacchi nel 1976 con il nome di White Knight sviluppato inizialmente in Pascal e poi portato su Assembly 6502 per Apple II e con esso vinse il Campionato Europeo di Scacchi per Microcomputer nel 1983, un anno dopo Cyrus. White Knight fu commercializzato in due versioni per BBC Micro ed Acorn Electron e la novità rispetto ai programmi analoghi era di calcolare e visualizzare la variante principale, la cosiddetta "Best Line", che in seguito divenne un tratto comune in tutti i programmi di scacchi.

L'algoritmo di White Knight fu usato da Bryant come base per lo sviluppo di tutta la serie denominata Colossus Chess dal 1983 in avanti, con titoli e versioni per un gran numero di piattaforme 8/16-bit degli anni 80, inclusi Amiga, Atari ST e IBM PC. Nel 1985 la rivista Zzap!64 decretò Colossus Chess il miglior programma di scacchi per home computer. Negli anni seguenti Bryant non smise mai di migliorare il suo motore alla base di Colossus Chess. Le ultime versioni risalgono al 2008 quando uscì "Colossus 2008b".

La sfida

Cyrus e Colossus sono entrambi programmi semplici da usare e con interfacce utente piuttosto intuitive (per i canoni di allora). Per cominciare a giocare contro il computer con il Bianco basta fare la prima mossa oppure accedere ai comandi avanzati per impostare il livello di gioco, giocare con il Nero, ruotare la scacchiera, impostare una posizione diversa da quella iniziale, ecc. In questo, Cyrus per ZX Spectrum è più intutivo presentando sullo schermo le opzioni di gioco cui si accede premendo i tasti alfabetici, mentre Colossus necessita della pressione della combinazione di tasti CTRL-<tasto> per attivare le varie funzioni e modalità di gioco. Entrambi i programmi offrono una visualizzazione grafica della scacchiera più che sufficiente per poter affrontare tutta una partita sullo schermo (all'epoca molti giocatori tenevano una scacchiera reale accanto al computer sulla quale replicare le mosse giocate).

Il match fra i due programmi è stato svolto impostando condizioni di parità per entrambi in quanto a forza di gioco, con un numero

uguale di secondi per mossa. Gli emulatori utilizzati per far girare i programmi (Spectaculator e Altirra su Windows) sono stati settati per ricreare le condizioni base dei due home computer: ZX Spectrum 48K e Atari 800XL. La sfida si è dipanata sulla distanza di 6 partite a cadenza fissa (30 secondi per mossa) e ciascun software ha giocato 3 volte con il Bianco e 3 volte con il Nero. Ecco l'esito dello scontro fra questi due storici programmi di scacchi: Colossus 4,5 - Cyrus 1,5. Colossus si è dimostrato molto più robusto ed ha mantenuto durante tutto il match uno stile guardingo ma solido. Cyrus è partito bene vincendo la prima sfida con il Bianco ma ha perso i 3 successivi incontri pur mostrando qualche idea originale per l'attacco e addirittura alcuni *escamotage* puramente tecnici per rimanere alla pari durante le partite. Entrambi i motori hanno deviato abbastanza presto dalle linee canoniche delle aperture che si sono trovati a giocare ma nel medio gioco ha prevalso la tecnica conservativa di Colossus. Le partite si sono prolungate per circa 40-50 mosse e l'unica patta è arrivata per ripetizione di mosse. Non avendo i motori la possibilità di abbandonare in caso di manifesta inferiorità di materiale o di posizione, si è provveduto a decretare manualmente l'esito delle partite (per inciso il software in vantaggio sarebbe comunque arrivato a dare scacco matto all'avversario). Tutte le 6 partite del match sono disponibili su richiesta in formato PGN.

di *Cercamon*

Conclusioni e giudizi

Cyrus IS Chess - Intelligent Software Ltd / Richard Lang - 1983 - ZX Spectrum 16/48K

GIOCABILITA'

75%

L'interfaccia utente prevede l'uso semplice della tastiera e la rappresentazione grafica è sufficiente per giocare solo con l'ausilio dello schermo. Il gioco è riprodotto in tutte le sue regole (presa *en-passant*, patta per ripetizione di mosse, regola delle 50 mosse, gestione dei finali, ecc.) e con i diversi livelli a disposizione costituisce un valido avversario ancora oggi (la forza di gioco è stimata intorno ai 1650-1750 punti ELO).

LONGEVITA'

90%

Per gli appassionati di scacchi non c'è teoricamente fine al numero di partite di buon livello che si possono disputare sul proprio ZX Spectrum. Il motore affronta con disinvoltura tutte le aperture e le relative varianti più giocate di sempre.

Colossus Chess 3.0 - English Software Company UK - 1984 - Atari 400/800 XL/XE

GIOCABILITA'

80%

Il programma propone un'ampia gamma di livelli di gioco, personalizzabili per creare un numero ingente di combinazioni. Una rappresentazione grafica essenziale ma efficace della scacchiera permette di seguire bene le mosse e la posizione raggiunta. Per accedere alle varie opzioni è necessario dare uno sguardo al manuale. Tutte le regole di gioco sono rispettate e la forza di gioco si attesta intorno ai 1750-1850 punti ELO.

LONGEVITA'

90%

Valgono le stesse valutazioni espresse per Cyrus, Colossus Chess era ed è un buon avversario per i giocatori amatoriali e non. Sicuramente uno dei migliori, se non il migliore, *chess engine* per Atari e per le macchine a 8 bit degli anni Ottanta, versatile e con un alto livello di personalizzazione.

Bibliografia

AlphaZero / DeepMind Paper - https://arxiv.org/pdf/1712.01815.pdf

Partite esemplari AlphaZero Vs Stockfish 8 - https://chess24.com/en/watch/live-tournaments/alphazero-vs-stockfish/1/1/1

Richard Lang - https://chessprogramming.wikispaces.com/Richard+Lang

Martin Bryant - https://chessprogramming.wikispaces.com/Martin+Bryant

Intelligent Software (Cyrus) - https://chessprogramming.wikispaces.com/Intelligent+Software

David Levy's Bet - http://www.computerhistory.org/chess/levys-bet/

Chess Programming - https://chessprogramming.wikispaces.com

RetroSpace: La conquista del cielo

di Marco Fanciulli

Missione: Retrospazio.

A partire da questo numero di RetroMagazine apriamo una nuova rubrica, **RetroSpace**. Articolo dopo articolo, incontreremo le personalità che hanno fondato la scienza e la tecnologia delle missioni spaziali, analizzeremo in dettaglio alcune delle macchine più famose e complesse e proveremo, nel nostro piccolo, a costruire delle riproduzioni di alcune di queste tecnologie nella speranza di poter vivere l'emozione di chi le ha progettate e guidate verso altri mondi.

Non abbiamo scopi enciclopedici né didattici né ci prefiggiamo scopi "alti" o particolarmente profondi ma, in un mondo che nel valore del passato costruisce le basi del proprio futuro, ci pareva sensato tra un videogioco e un retro-computer parlare anche delle tecnologie e degli uomini che ci hanno permesso di superare i limiti dell'atmosfera per avventurarci in un universo pericoloso e poco amico. Un universo che però è nel nostro destino studiare e esplorare.

Prima di addentrarci nella storia e nei dettagli si singoli satelliti e sonde, di missioni e di computer ai quali dobbiamo molto del nostro presente, ci faremo le ossa con un po' di storia e qualche fondamentale equazione che ci accompagnerà nella comprensione di quello che seguirà.

Per quanto possa sembrare strano e contro-intuitivo, alcune delle macchine più complesse che l'Uomo abbia mai costruito e la scienza che le ha permesse muovono da poche equazioni, tutto sommato semplici da comprendere (adesso che sono state spiegate!).

È con questo spirito di scoperta che a partire da oggi e molto di più nei prossimi cinque o sei articoli, scopriremo come abbiamo inviato in orbita satelliti di ogni genere, quali difficoltà abbiamo dovuto affrontare per mantenerne il controllo e ricevere da essi informazioni preziose e per divertirci un po' metteremo in piede con meno di 30 euro una stazione di ricezione che sarà il punto di partenza per lo sviluppo di un nostro retro-satellite da spedire in orbita (burocrazia e generosità permettendo).

Gli albori dell'astronautica

La fascinazione del cielo stellato ha accompagnato tutti gli esseri viventi, uomini e animali, da sempre. La volta celeste, così apparentemente vicina e al tempo stesso intangibile, ha rappresentato un elemento di mistero e sgomento e al tempo stesso un polo di fascinazione incontenibile per un essere curioso come l'Uomo. Che cosa sono quei puntini luminosi che scintillano di notte? Perché alcuni di quei puntini non scintillano e sembrano muoversi di moto casuale e erratico nel cielo senza rispettare il movimento di tutto il resto? Che cosa sono quel faro accecante che ci scalda di giorno e quel disco che ci rischiara pallidamente di notte?

Come spesso accade di fronte all'ignoto, l'Uomo si è dato da prima una risposta mistica o fideistica che non necessitasse di spiegazioni fattuali ma che desse un nome alla paura al fine di esorcizzarla. Poi, grazie a menti più speculative e curiose, ha iniziato a riconoscere la regolarità nell'irregolarità e a costruirsi da solo gli strumenti del sapere, fino a realizzare che se non esiste alcuna cupola sulla quale sono incastonati quei punti brillanti; nel corso dei secoli abbiamo capito che si tratta di corpi celesti che, per quanto lontani, sono teoricamente raggiungibili. La consapevolezza di una realtà cosmologica e della sua raggiungibilità ha dato il via al più grande sviluppo di conoscenza e tecnica che l'uomo abbia mai conosciuto. Dalle teorie teogoniche e da quelle geocentriche e meccanicistiche siamo passati a una concezione dell'universo che contempla l'infinito e la marginalità del nostro ruolo; la nostra piccolezza ci ha invitati a perseguire la grandezza, dando prova di un costante progresso del pensiero matematico e astratto e di un'incredibile capacità di realizzazione degli strumenti volti a progredire raggiungere i corpi celesti vicini, sognando di arrivare a quelli lontani.

Era solo questione di tempo prima che, volgendo gli occhi al cielo e avendo compreso che ciò che vedeva non era l'illusione di una volta dipinta, desiderassimo esplorare lo spazio. Non per mera asserzione di potere ma perché è necessario farlo per dare una chance all'umanità di trovare altri luoghi in cui proliferare. O forse sopravvivere a se stessa.

Il sogno di conquistare la Luna e i pianeti è vecchio quanto il momento in cui il genere umano si è reso di non trovarsi al cospetto di sferette dipinte su un'enorme cupola ma di veri e propri corpi celesti. In parallelo col procedere delle scoperte scientifiche e alla rappresentazione fisica e matematica della realtà, la letteratura si è sbizzarrita nell'immaginare i modi e i tempi di queste conquiste. Luciano di Samostata, scrittore greco del II secolo d.C. nel suo "Storia Vera" immaginava il viaggio verso la Luna a bordo di una nave che spinta da un vento fortissimo si staccava dalle acque come una foglia dall'albero. Dante offre sfoggio di sapienza nel secondo canto del Paradiso nel quale sviluppa un vero e proprio trattato delle conoscenze seleniche dell'epoca, spingendosi fino a dare una interpretazione fisica della natura dei mari (anche se poi Beatrice redarguirà Virgilio riportando il tutto su un piano più teologico e metafisico). Nell'"Orlando Furioso" l'Ariosto invia Astolfo sulla Luna a cercare il senno perduto di Orlando, impazzito in seguito al tradimento di Angelica. Il viaggio, in questo caso, si svolge per mezzo di un carro trainato da cavalli (il carro di Elia) ma quadrupedi di fuoco, così come di fuoco è la scia che lascia. Nell'evidente insostenibilità di un carro volante che stride con la pesantezza del suo trascinarsi lungo le strade sterrate dell'Italia che fu, il poeta unì la simbologia mistica all'esigenza di dare una "spinta in più" al mezzo di trasporto affinché fosse verosimile che potesse levarsi in cielo per raggiungere una meta altrimenti troppo remota. Sarà comunque Jules Verne a dare una prima, impressionante, descrizione di un viaggio spaziale in termini più vicini a quelli moderni. In "Dalla Terra alla Luna", lo scrittore

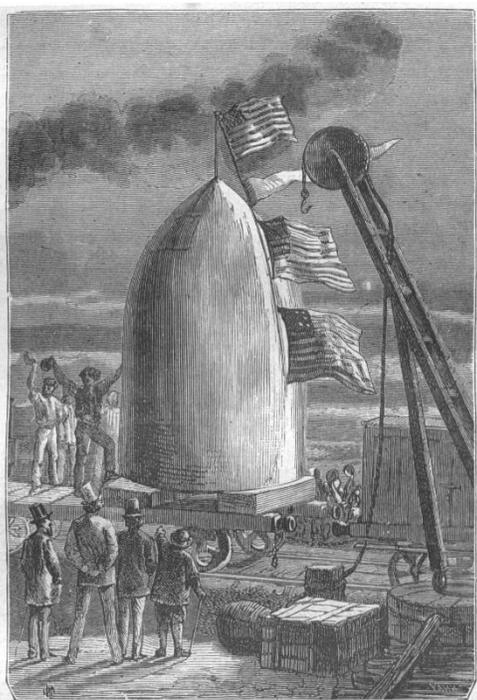
immagina il volo balistico verso la Luna di Impey Barbicane, del capitano Nicholl e di Michel Ardan calcolando al meglio delle conoscenze disponibili la durata e le caratteristiche del volo. Per un errore di calcolo il proiettile, sparato con un mega-cannone al fulmicotone, non raggiungerà mai la superficie della Luna ma vi entrerà in orbita stabile mettendo a rischio la vita dei tre occupanti, presto a corto di cibo e aria. Sarà la loro intelligenza a permettergli di uscire dall'orbita e a fare ritorno sulla Terra, ammarando in tutta sicurezza esattamente come fanno le sonde moderne. Come avremo modo di questa modalità di rientro è stata data tutt'altro che per scontata fino a buona parte degli anni '50.

L'arrivée du projectile à Stone's-Hill (p. 139).

Fig. 1: il proiettile di Jules verne

Non c'è bisogno di ampliare l'elenco della letteratura fantastica per cogliere che questo spaccato temporale mostra come l'evoluzione tecnologica abbia contribuito a creare scenari di conquista dello spazio sempre più verosimili, fino al punto in cui fantasie più vicine alla metafisica e alla filosofia che non alla scienza e alla tecnica si sono trasformate prima in consapevolezza, poi in un contesto proprio e quindi in un problema determinato al quale dare una precisa risposta: come acquisire una spinta sufficiente a lasciare la superficie del pianeta senza finire dilaniati nel tentativo? Come Jules Verne aveva così ben tradotto in parole e fantasia, per andare nello spazio ci serve un proiettile in grado di spingersi autonomamente: un razzo.

Il razzo

È ben documentato e largamente condiviso nella comunità degli storici della scienza, che l'idea del razzo abbia la propria origine nella Cina dell'XI secolo. Il manoscritto "**Wu Cling Tsung Yao**", redatto nel 1040 d.C., documenta come all'inizio del nuovo millennio i cinesi conoscessero già l'uso della polvere da sparo e ne avessero definito in modo puntuale e rigoroso il processo di preparazione e i calcoli per determinarne la potenza in relazione agli usi. Informazioni aggiuntive e prove più dirette ci vengono tramandate da documenti di due secoli dopo, precisamente nelle cronache dell'assedio di **Kai-fung-fu** del 1232 durante il quale le truppe cinesi si difesero e sconfissero le schiere dei mongoli lanciando su di loro una tempesta di "frecce di fuoco". I disegni e i dipinti del periodo ritraggono queste frecce non come semplici dardi ai quali era stata applicata una benda imbevuta di pece infuocata ma come aste alle quali era stata legata una sorta di bomba-razzo; questo razzo produceva sia una spinta addizionale, favorendo un volo più rettilineo e a lunga gittata, sia la capacità – esplodendo - di spargere elementi incendiari devastanti su un'area piuttosto vasta.

Se è vero che l'uso documentato del razzo è riconducibile a queste testimonianze, è altrettanto lecito presupporre che i cinesi avessero già introdotto sistemi bellici basati sullo stesso principio. La storia dei giochi pirotecnici, le cui competenze sono assimilabili, precede questi scritti documentali; già nel VII secolo i Bizantini avevano prodotto una sorta di razzo usato per spingere a distanza una lancia di bambù dopo aver introdotto l'uso della polvere da sparo portata dai Saraceni, i quali l'avevano ricevuta proprio dai cinesi. Risalendo ancora più indietro nel tempo, la polvere da sparo o meglio il salnitro che ne è un costituente, veniva chiamato dagli egiziani "neve della Cina" a riassumere una storia molto più risalente e un'origine inequivocabile. Fatto sta che nella seconda metà del 1200 il "principio del razzo" aveva iniziato a diffondersi dal medio-oriente nell'Europa continentale. Dopo un primo interesse per fini militari, il suo uso per tali scopi scemò mano a mano che le armi da fuoco rese possibili dalla diffusione della polvere da sparo si sviluppavano e perfezionavano; dopo neanche un secolo, il razzo fu relegato alle sole finalità pirotecniche almeno fino ai primi decenni del 1800 quando Sir William Congreve, colonnello dell'esercito della Regina distaccato in India, sviluppò un razzo a combustibile solido di scarsa precisione ma usato con profitto nell'attacco alla città di Boulogne e soprattutto nell'attacco a Copenaghen, nel quale l'armata inglese lanciò oltre 40.000 razzi devastando la città. Il razzo di Congreve fu utilizzato regolarmente durante le guerre napoleoniche, soppiantato solo dopo il 1850 da una sua evoluzione a opera di William Hale che sostituì l'asta guida (che lo rendeva simile ai razzi cinesi del 1200) con gli ugelli che oggi siamo abituati a vedere.

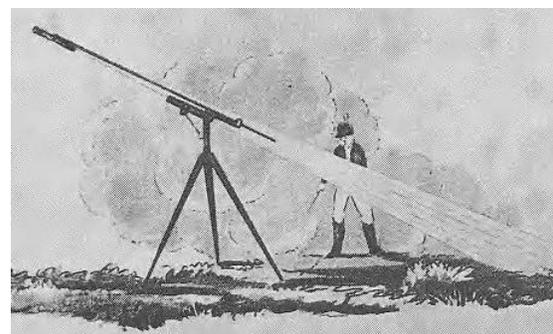



Fig. 2: il razzo Congreve

Non ci stupisce affatto che l'origine del razzo sia riconducibile a sviluppi bellici. Come spesso è accaduto, la ricerca tecnologica e scientifica è stata mossa da interessi di natura militare e solo successivamente ha trovato applicazione in sede civile. La storia degli antesignani dei razzi non ha fatto eccezione La stessa storia dell'astronautica è soprattutto la storia di un inseguimento tecnico tra gli USA e l'URSS in un'escalation militare senza precedenti. La corsa delle due super-potenze a non lasciare spazi di manovra non presidiati al nemico, ha prodotto una delle più rapide evoluzioni tecnologiche mai viste. Per quasi tutti i primi sessanta anni di questa rincorsa, gli URSS hanno avuto una leadership apparentemente netta ma in realtà costellata di sacrifici.

Fa quindi un certo contrasto filosofico il fatto che la storia dell'astronautica sia entrata nell'era moderna grazie al genio di un

sognatore che era quanto di più lontano dallo spirito bellico si possa immaginare. In un mondo dominato da un montare di opportunità di guerra che da lì a pochi decenni sarebbero deflagrate nella Prima Guerra Mondiale, nel gelo della tundra siberiana, si accese un faro.

Era inverno a Iževskoe, una piccola cittadina a 150 km da Mosca. Il ghiaccio di uno dei mesi di dicembre più ingenerosi della storia aveva imprigionato gli abitanti in un isolamento forzoso all'interno delle proprie case. Tra questi prigionieri della natura, un ragazzino di tredici anni che era appena uscito da una storia personale di malattia e della perdita della madre. Non una storia infrequente nella povertà diffusa della steppa siberiana. Il piccolo Konstantin aveva calcato per pochi anni il suolo dell'Impero Russo ma di chilometri ne aveva già percorsi molti. Il padre aveva cambiato spesso lavoro e con esso città; la famiglia, una moglie e tredici figli, lo avevano seguito tra alterne vicende e ricorrenti povertà. A aggiungere danno alla beffa un'epidemia di scarlattina non risparmiò il giovane, che all'età di appena 10 anni aveva ricevuto in eredità una sordità pressoché totale e gravemente invalidante che, nonostante la sua spiccata e vivace propensione alle scienze naturali, lo privò della possibilità di frequentare la scuola con profitto. Persa anche la madre, Konstantin Tsiolkovskij (nella traslitterazione anglosassone, Ciolkoski in quella italiana) si rifugiò e trovò conforto nello studio da autodidatta, sfruttando la modesta ma valida biblioteca di casa. Per molti versi, questa fu la sua fortuna. Costretto ad accontentarsi di ciò che aveva e non potendo contare su un tutore che lo aiutasse a comprendere la meccanica delle cose che studiava, il piccolo Konstantin sviluppò un grande senso critico e s'ingegnò di cercare da solo le spiegazioni ai fenomeni più complessi sviluppando una creatività scientifica che ha fatto da punto di riferimento per lo studio, decenni dopo, delle dinamiche del pensiero creativo e laterale.

Il padre Eduard, pur nei limiti di una disponibilità economica ridotta, riconobbe il talento del figlio e s'ingegnò di inviarlo a Mosca dove avrebbe potuto accedere a un sapere più ampio. Non che potesse permettersi studi propriamente detti, anzi! Mosca, per Konstantin, significò soprattutto il poter attingere agli innumerevoli libri di una biblioteca più ampia, quella del museo Rumjancev presso la quale ampliò il proprio campo di interesse alla fisica, alla matematica e alla biologia.

Questo ragazzino sveglio e apparentemente isolato dal mondo che lo circondava destò l'attenzione di Nikolaj Fëdorov, il fondatore della filosofia del "cosmismo" (una corrente filosofica che vedeva nell'uomo un fattore positivo e determinante per l'evoluzione del cosmo) e personalità illustre della cultura russa. Il "Socrate di Mosca", come fu chiamato successivamente, provvide a dare ordine agli studi del piccolo Tsiolkovskij, indirizzandolo in modo sistematico e progressivo verso discipline più tecniche e strutturate. Furono tre anni di maturazione repentina e soprattutto gli anni nei quali Konstantin, stimolato da Fëdorov, iniziò a concepire la possibilità per il genere umano di abbandonare la Terra per conquistare lo spazio.

In un mondo che aveva appena iniziato a sognare di poter volare come gli uccelli, questo ragazzino sordo e scavato dalla fame si era dato l'obiettivo per la vita: lasciare il pianeta, sfruttando una forma di energia per spingere una navicella fuori dall'orbita così come la forza centrifuga allontana il sasso lanciato dalla fionda. Era nata l'idea del razzo come vettore.

I primi passi

La piattaforma inizia a vibrare. Il rumore sordo delle pompe che attraverso i cordoni ombelicali tengono in pressione il propellente nei serbatoi lasciano il posto a cinque esplosioni contemporanee; cinque colpi violenti che paiono uno solo. Siamo a -8.9 secondi dalla fine del count-down.

Quello che era un debole vapore si trasforma in un inferno di fuoco mentre il rumore sale altissimo e le vibrazioni squassano il terreno per chilometri.

Il gigante bianco e nero si solleva di pochi cm mentre urla e combatte contro la forza di gravità e il proprio stesso peso. Il conto alla rovescia è finito. Il segno meno è scomparso, il tempo scorre nella direzione del futuro.

Un secondo e mezzo. Questo obelisco tecnologico, monumento alla grandezza dell'ingegno umano, ha percorso pochi metri e si inclina impercettibilmente di 1,25° per allontanarsi dalla torre di lancio. È ancora praticamente fermo, basterebbe un alito di vento per spingerlo contro la struttura di tralicci. Ruggendo come un leone arranca verso l'alto, acquisendo velocità.

Adesso sono passati dieci secondi, appena dieci secondi. Il Saturno V ha percorso solo 137 metri, una distanza di poco superiore alla propria lunghezza, ma sufficiente a superare l'altezza della torre di lancio; si raddrizza inclinandosi al contrario di 1,25° e inizia la propria galoppata verso il cielo.

In quei dieci secondi il razzo ha perso il 4% della propria massa consumando 140 tonnellate di propellente.

Nulla di quanto descritto è lasciato al caso: tutto risponde a precise equazioni, tutto è previsto e tutto è programmato. Tutto è scienza.

Fig. 3: Il Saturno V al decollo. Sono passati 9 secondi

Il principio di funzionamento di un razzo è oggi quasi intuitivo: un oggetto solido (il nostro veicolo) produce una spinta espellendo ad alta velocità un altro materiale (un gas, per esempio) il quale, per il principio di conservazione della quantità di moto previsto dalla seconda legge della dinamica, produce nell'oggetto una spinta in direzione opposta che determina una quantità di moto uguale a

quella del gas espulso (in condizioni ideali). Va da sé che tanto più grande è questa velocità di uscita e tanto maggiore sarà la spinta che viene generata, tenendo in considerazione le masse relative del razzo e del gas espulso.

Questo principio, enunciato Isaac Newton nel secondo principio della dinamica, aveva già chiarito la natura della spinta che un fuoco pirotecnico riceveva in direzione contraria a quella di espulsione del combustibile e aveva permesso di calcolare la quantità di propellente necessario a raggiungere altezze calibrate. Ciò che non era ancora noto, ai fini di un volo controllato, era la relazione intercorrente tra tutti gli elementi in gioco (le masse, le pressioni, la gravità, la rigidità del vettore, ...) e quindi la capacità di calcolare le quantità di ciascuno di essi per spingere il razzo al di fuori dell'abbraccio gravitazionale terrestre. Anzi, l'idea era talmente balzana da non essere presa in considerazione per un altro paio di secoli.

Fig. 4: Il principio di azione / reazione

La conquista della "formula del razzo" non fu un fatto subitaneo ma un percorso progressivo, secondo un modello sistematico di acquisizione di conoscenze puntuali e essenziali sulle quali si sono fondate le conoscenze successive.

Dopo un triennio di proficui studi e di formazione a Mosca, Tsiolkovskij tornò nella sua Iževskoe e dopo poco ottenne la cattedra in una piccola scuola di Borovsk, cattedra che tenne per 12 anni al fine di mantenere la famiglia (ebbe sette figli... le notti siberiane sono lunghe e fredde!) e potersi permettere un piccolo laboratorio privato nel quale condurre le proprie ricerche in totale autonomia. In quegli anni, seguendo un interesse diffuso nell'ultima parte del 1800, s'interessò soprattutto di dinamica dei gas e di aeronautica, con specifico interesse alla progettazione di un dirigibile a pallone metallico che avrebbe risolto i problemi di infiammabilità dei materiali ma soprattutto gli avrebbe consentito di approfondire gli studi sugli effetti dell'atmosfera sulla dinamica del volo. In quegli stessi anni, com'è emerso dallo studio dei suoi appunti, giunse alle medesime conclusioni e equazioni sviluppate dai fratelli Wright negli USA anche se non nutrì mai un vero interesse per il volo in atmosfera. La finalità di Tsiolkovskij era quella di creare i modelli di calcolo utili alla determinazione degli effetti dell'aria sul moto di proietti a propulsione autonoma e alle condizioni dello spazio esterno, studi che gli permisero di teorizzare per primo l'assenza di gravità e le sue ripercussioni sulla dinamica dei fluidi (in particolare del sangue nei sistemi meccanici biologici altrimenti detti... noi!). È di questi anni un suo disegno di una navicella spaziale, la cui organizzazione è incredibilmente simile all'architettura delle capsule degli anni '50.

Nel 1896 questo campo di studi ormai costituiva il suo unico, ossessionante obiettivo scientifico. Riprese le idee di gioventù e, forte di una conoscenza più ampia e dell'aver determinato le equazioni necessarie a completare il quadro, si dedicò allo sviluppo degli elementi matematici necessari per razionalizzare le variabili della propulsione a razzo per i viaggi interplanetari. Appena un anno dopo sviluppò l'equazione che finalmente mise in relazione la velocità del razzo, la sua massa a vuoto e a pieno carico, nonché la massa del combustibile utilizzato e la velocità di espulsione del gas combusti. L'equazione prese il nome di **Equazione del razzo di Tsiolkovskij** ed è ancora oggi il punto di partenza dell'astronautica e dell'astrodinamica moderne, dalle V2 tedesche al recentissimo Falcon Heavy della SpaceX.

Una legge per governarli tutti

Nei pochi minuti che sono trascorsi dalla lettura del paragrafo che ha visto il decollo del Saturno V, il razzo ha percorso una settantina di chilometri accelerando continuamente mentre espelle il propellente combusto sotto forma di gas. Questo bestione dal peso difficilmente valutabile a mente, è passato da un moto lento e arrancato a circa 10.000 km/h e, come previsto dagli ingegneri, ha consumato poco più di due milioni di kg di propellente.

La previsione degli ingegneri è figlia legittima e diretta dell'equazione del razzo di Tsiolkovskij la quale recita che

$$\Delta v = v_e ln \frac{m_i}{m_f}$$

dove

Δv è la variazione di velocità del razzo

v_e è la velocità equivalente (pari all'impulso istantaneo dato dal motore moltiplicata per l'accelerazione di gravità al livello del mare)

m_i la massa del razzo a vuoto

m_f la massa del razzo con il carico di propellente

L'equazione considera il fatto che la massa del razzo diminuisce a mano a mano che la combustione e propulsione procede poiché il propellente viene consumato e espulso alleggerendo il vettore. In particolare, l'equazione ci dice che l'incremento della velocità del razzo dopo l'inizio della combustione (Δv) è proporzionale alla velocità equivalente di uscita dei gas dal propulsore (v_e) e al logaritmo del rapporto tra la massa del razzo al decollo e la massa a vuoto. Tanto più è leggero il razzo, tanto più a parità di spinta andrà velocemente. Come potete notare, ho detto che l'equazione

considera esclusivamente la velocità equivalente e non la velocità effettiva. Infatti, in atmosfera le due velocità differiscono poiché il razzo risente nel suo avanzamento della differenza di pressione tra la pressione atmosferica e quella data dall'uscita del gas dal propulsore.

Che cos'è questa "velocità equivalente"? La velocità equivalente è data dal cosiddetto Impulso Specifico (I_{sp}) moltiplicato per l'accelerazione gravitazionale al livello del mare (g_0) e la sua forma diventa quindi:

$$\Delta v = I_{sp}\, g_0 ln\frac{m_i}{m_f}$$

L'impulso specifico è una misura dell'efficienza di un motore a razzo, ovvero quanto efficientemente il razzo nel suo complesso converte il propellente in spinta. Da un punto di vista tecnico è l'impulso totale (o meglio, il cambiamento nel momento) prodotto per ciascuna unità di propellente consumata ed equivale alla spinta generata divisa per il flusso di propellente espresso in termini di massa o di peso. A seconda che venga considerata la massa (espressa in chilogrammi) o il peso (espresso in newton o libbre), l'I_{sp} è una misura di velocità o di tempo. Moltiplicando il rateo del flusso per la gravità al livello del suolo, l'I_{sp} viene convertito da misura di massa a misura di peso, rendendo di fatto l'equazione utilizzabile senza modifiche sia con il sistema metrico che con il sistema imperiale.

Va da sé che le forze che si applicano al razzo nel suo moto si modificano continuamente durante il volo. La massa del razzo varia a mano a mano che il combustibile viene consumato e espulso, la densità atmosferica e le relative pressioni e attrito cambiano a seconda dell'altezza e delle condizioni meteorologiche, gli effetti stessi del consumo di carburante spostano le masse residue all'interno dei serbatoi alterando il centro di massa, e così via. L'equazione classica si riferisce a un razzo ideale in condizioni ideali ma è semplice, a titolo di esempio, prendere in considerazione gli effetti gravitazionali modificando l'equazione in

$$\Delta v = I_{sp}\, g_0 ln\frac{m_i}{m_f} - g_0 sa$$

dove g_0 è l'accelerazione di gravità al livello del mare e sa è la spinta attiva, vale a dire la durata dell'accensione in secondi.

Questa equazione è di fondamentale importanza per determinare le caratteristiche stesse del razzo e l'efficacia e efficienza del suo sistema propulsivo. Come abbiamo detto, l'impulso specifico I_{sp} è una misura dell'efficienza della specifica combinazione di schema di propulsione (che considera anche l'ampiezza dell'area complessiva degli ugelli), velocità del flusso in uscita e variazione nella massa del carburante (il consumo, diremmo in modo colloquiale). Queste caratteristiche variano profondamente a seconda del tipo di combustibile e di comburente e delle caratteristiche strutturali del razzo stesso. Alcuni tipi di propellente richiedono una struttura più rigida e pesante e quindi il rapporto tra la massa del razzo a vuoto e quella del razzo con il combustibile varia di conseguenza. Ad esempio, la massa di una struttura non riutilizzabile (i razzi a perdere che bruciano in atmosfera dopo l'uso) costituisce almeno il 6% del totale della massa quando si usa la combinazione kerosene / idrogeno liquido ma almeno il 9% della massa totale quando si usa la combinazione idrogeno liquido / ossigeno liquido, grazie al fatto che l'idrogeno ha una densità molto più bassa del kerosene. In termini pratici, più è alto il rapporto tra massa totale e massa a vuoto e tanto più pesante può essere il razzo a parità di prestazioni e quindi più grande il carico utile che può trasportare. Tanto per darci un termine di paragone, in un aereo come il Concorde o il suo clone sovietico Tupolev 144, la massa a vuoto rappresenta poco meno del 45% della massa totale. Il rapporto tra la massa a vuoto e la massa totale è chiamato "**indice di struttura**" o "indice strutturale".

Per semplicità nella formulazione dell'equazione del razzo che abbiamo illustrato, l'I_{sp} è considerata costante ma in realtà introducendo i fattori di resistenza dell'aria e di altre forme di attrito perde efficacia per un valore compreso tra il10% e il 20% a pressione atmosferica. Intuitivamente è ovvio che fuori dall'atmosfera, venendo meno questi elementi di contrasto, il razzo è più efficiente.

Combustibili differenti e indici di struttura differenti producono I_{sp} differenti. A ciascun profilo di missione e in relazione ai vincoli di costo corrisponde una combinazione di propellente e conseguente indice di struttura massimo. Le tecnologie attualmente più diffuse per i propellenti sono:

Tipo di propellente	$I_{sp}(s)$	Velivoli equipaggiati
Alluminio – Perclorato di ammonio	270	Razzi a combustibile solido (Ariane V, booster laterali dello Space Shuttle)
UDMH – Perossido di azoto	290	Propellente liquido in tutti gli Ariane da I a IV
UDMH – Ossigeno liquido (HMDH/LOX)	310	Propellente liquido nei razzi Proton
Kerosene – Ossigeno liquido (RP/LOX)	330	Propellente liquido nei razzi Atlas, Delta, Soyuz, Proton, Zenith e Falcon 9
Idrogeno Liquido – Ossigeno liquido (LH2/LOX)	450	Propellente liquido nei propulsori dello Space Shuttle (serbatoio centrale), Ariane V, Falcon 9 e Falcon Heavy

Determinare gli indici di struttura massimi non è stata un'impresa semplice.

Negli anni immediatamente successivi allo sviluppo dell'equazione del razzo e prima della sua pubblicazione, Tsiolkovskij proseguì i propri studi apparentemente distraendosi dai temi aerospaziali per concentrarsi su sviluppi di tipo aeronautico. Come abbiamo già detto, sviluppò le equazioni necessarie allo sviluppo di un modello di dirigibile metallico e di un aeroplano che riteneva essenziali per ottenere dati più precisi sulla resistenza dell'aria e le altre componenti dinamiche del volo in atmosfera. Il suo scopo, pur raccogliendo l'interesse dell'Accademia delle Scienze, era finalizzato principalmente a acquisire gli elementi mancanti per estendere in modo corretto la portata della propria equazione del razzo.

Con mezzi proprio costruì la prima galleria del vento mai realizzata in Russia e grazie a questa consolidò i risultati delle proprie ricerche, ottenendo i fondi necessari a costruirne una ancora più grande e in grado di simulare il volo a velocità ancora fuori dalla portata dell'industria aeronautica.Nel 1903 Tsiolkovskij dette finalmente alle stampe "L'esplorazione dello spazio cosmico per mezzo di motori a reazione", la sua opera più importante nella quale usciva allo scoperto formulando l'equazione del razzo e i calcoli necessari a dimostrare la realizzabilità del volo spaziale, le velocità critiche necessarie per ottenerlo e la quantità di propellente necessario.

Fig. 5: Konstantin Ėduardovič Tsiolkovskij nel suo laboratorio privato

Facciamo un esempio. Per raggiungere un'orbita bassa (diciamo a 200km di altitudine), un mezzo deve raggiungere una variazione di velocità Δv pari a 7800 m/s (in realtà considerando anche gli effetti degli attriti e della gravità dovremmo aumentarla di altri 1700 m/s). Partendo dall'equazione del razzo, possiamo derivare che la frazione della massa totale prima del decollo che è rappresentata dal propellente è data dall'equazione

$$\frac{massa\ del\ propellente}{massa\ totale\ a\ pieno\ carico} = \frac{mf - m_i}{m_f} = 1 - \frac{m_i}{m_f}$$

Se risolviamo l'equazione del razzo per questo rapporto, otteniamo che

$$1 - \frac{m_i}{m_f} = 1 - e^{-\frac{\Delta v}{v_e}}$$

che rappresenta la frazione della massa totale del razzo che deve essere il combustibile necessario per raggiungere quella variazione di velocità.

Volendo raggiungere 200km di altitudine con il nostro razzo dobbiamo far sì che il propellente costituisca l'85% della massa totale del veicolo.

$$1 - \frac{m_i}{m_f} = 1 - e^{-\frac{7800}{4000}} = 0.85$$

Spezzatino spaziale

Sono trascorsi tre minuti dal momento in cui il Saturno V ci ha lasciati con il naso all'insù a seguire la sua scia di fuoco. Pochi istanti fa ha raggiunto il picco di pressione aerodinamica pari a circa 33,5 KPa. Dopo essere stato sottoposto a forze enormi che hanno rischiato di ridurlo in pezzi, le cose si sono fatte meno preoccupanti per i tre uomini a bordo. L'atmosfera è meno densa a questa altitudine e per l'enorme razzo è arrivato il momento di ravvivare la propria spinta. In uno sbuffo di carburante i cinque motori si spengono e per un attimo la scia del razzo diventa scura e il rumore cessa. Solo per un attimo sembra morto, un proiettile lanciato senza controllo e destinato a essere sopraffatto dalla gravità. È solo il timore di un attimo perché in pochi istanti uno sbuffo di carburante e una serie di cariche esplosive fanno distaccare il primo stadio del razzo. Ancora due secondi per permettere al moncone di allontanarsi un po' e poi, in un lampo di luce, i razzi del secondo stadio si accendono e iniziano a ruggire.

Adesso che il tempo è bello fuori dall'abitacolo dell'angusta navicella che ospita gli astronauti, la torre che li protegge viene espulsa e finalmente il pilota del modulo di comando e del modulo lunare, che siedono vicino ai due oblò, possono godersi lo spettacolo di un cielo che diventa sempre più nero in pieno giorno, mentre la curvatura della Terra da appena accennata inizia a disegnare un arco sempre più chiuso e definito.

In questi pochi minuti il Saturno V ***ha perso l'80% della sua massa iniziale*** *per arrivare issarsi fino a qui e ha ridotto la propria lunghezza di 42 metri. Ma non la propria spinta che invece aumenta, fornendo al razzo una buona parte della velocità mancante per sfuggire all'abbraccio della gravità.*

Fig. 6: Lo stadio S-IC si separa dal Saturno V. La spinta adesso è affidata allo stadio S-II

L'equazione di Tsiolkovskij mostrò quasi subito un limite molto grande all'uso di razzi a singolo stadio per l'invio di mezzi nello spazio. A mano a mano che il propellente viene espulso, infatti, la massa relativa del razzo stesso cresce rispetto al totale e se è vero che i fattori di resistenza atmosferica si riducono al salire di altitudine, è anche vero che il rapporto tra massa residua e spinta diventa sempre più sfavorevole.

L'indice di struttura (che come abbiamo visto indica il rapporto tra la massa del veicolo in sé e quella del veicolo col pieno di propellente e il suo carico utile) trova un limite superiore a seconda dell'I_{sp} del propellente utilizzato. In particolare, un razzo a cherosene-ossigeno liquido non può avere un indice di struttura superiore a 5,30% e un razzo a idrogeno liquido / ossigeno liquido (LH2/LOX) non può eccedere un indice di struttura di 11,5%. Abbiamo appena visto che per arrivare in orbita bassa (LEO, Low Earth Orbit) abbiamo bisogno che l'85% della massa del nostro razzo sia rappresentata dal carburante e per un propellente RP-LOX, come quello usato dal primo stadio del modernissimo Falcon 9, superiamo l'indice di struttura massimo arrivando oltre il 6%: non possiamo raggiungere l'orbita. Quand'anche rientrassimo esattamente al 5,30% la frazione di carico utile del nostro razzo sarebbe molto, molto piccola. Del tutto inutile.

La durezza dei numeri che escono dall'applicazione dell'equazione del razzo indicò molto presto che per aumentare il carico pagante di una missione spaziale si doveva cambiare la strategia di lancio, per evitare di doversi trascinare dietro quella parte di massa non più utile ai fini del raggiungimento dell'orbita. In sostanza, si doveva realizzare un razzo composto da più stadi ciascuno dei quali, terminata la propria funzione, poteva essere eliminato riportando l'I_{sp} a un valore più elevato. Questo approccio è essenziale per il volo spaziale e non è un caso che anche lo Sputnik, il primo satellite artificiale della storia, sia stato portato in orbita da un razzo a due stadi.

L'equazione del razzo si adatta facilmente allo scenario multistadio, tenendo in considerazione le masse e i diversi I_sp per ciascuno stadio.

Ad esempio, per un razzo a due stadi la formulazione generale diventa:

$$\Delta v = I_{sp1}\, g_0 \ln\left(\frac{M_T}{M_{s1}+M_2+M_U}\right) + I_{sp2}\, g_0 \ln\left(\frac{M_2+M_U}{M_{s2}+M_u}\right)$$

dove $M_T = M_1 + M_2 + M_U$

M_T è la massa totale del razzo sulla sua piazzola di lancio, incluso il carico pagante e il propellente

M_U è la massa del carico pagante (il satellite, gli astronauti, ecc)

M_1 è la massa totale del primo stadio

M_2 è la massa totale del secondo stadio

M_{S1} è la massa a vuoto del primo stadio

M_{S2} è la massa a vuoto del secondo stadio

I_{sp1} è l'impulso specifico del primo stadio

I_{sp2} è l'impulso specifico del secondo stadio

Questo è un articolo divulgativo e non faremo tutti i calcoli prendendo un razzo a esempio; vi basti sapere che decenni di dati quantitativi derivati dai razzi e ai veicoli spaziali la cui storia e analisi di dettaglio vedremo in futuro, hanno evidenziato una combinazione ottimale nella quale il secondo stadio è più leggero da quattro a cinque volte rispetto al primo stadio. Non stupisce quindi che il lancio col carico pagante più massiccio mai portato in orbita dal Saturno V, lo sfortunato Skylab, abbia visto un primo stadio da 2.286 tonnellate metriche di massa e un secondo stadio da 493 tonnellate metriche. Ai limiti della capacità di spinta del colosso di Von Braun e ai limiti dell'indice di struttura. Una macchina davvero meravigliosa.

Avendo I_{sp} diversi a seconda del propellente, ma anche massa differente, molto spesso i razzi multi-stadio utilizzano un primo stadio con propellente a alto indice di spinta e basso indice di struttura, per poi passare a un secondo stadio con un indice di spinta che cresce proporzionalmente grazie al più alto indice di struttura. Questa combinazione permette di rendere più efficiente la spinta del razzo e di accrescere la quota di massa dedicata al carico pagante. Laddove il Saturno V poté portare in orbita bassa un carico pagante di 116 tonnellate metriche (il 4% della propria massa), un moderno razzo a due stadi RP/LOX e LH2/LOX riesce a portare il 6% della propria massa. Un miglioramento che nel tempo è derivato dalla maggiore efficienza della tecnologia dei motori (+2% dal 1970!) e grazie all'equazione di Tsiolkovskij che ne ha indirizzato la ricerca!

A destinazione

Il secondo stadio del Saturno V sta spingendo con forza e costanza e si prepara a terminare il proprio compito. Mancano un minuto e trenta secondi alla separazione del secondo stadio e il motore centrale viene spento per rendere stabile e controllato il consumo di carburante.

Quando sono trascorsi circa 8 minuti e 30 secondi dal decollo il razzo varia il rapporto tra ossigeno e idrogeno per compensare gli squilibri di utilizzo che si sono verificati durante il volo a causa della diversa espansione dei due gas nei serbatoi. Da un rapporto 5:5:1 tra ossidante e carburante, passa a una miscela più ricca del secondo: 4:8:1. La spinta ne risente un po' ma la massa fortemente ridotta compensa abbondantemente contribuendo a mantenere un I_{sp} pressoché invariato. Gli astronauti non se ne accorgono neanche.

A 45 secondi dalla separazione il primo sensore che rileva l'approssimarsi della fine del propellente si attiva e l'elettronica si mette in attesa del secondo sensore per iniziare la procedura di spegnimento controllato. 9 minuti e 12 secondi dopo il decollo, il secondo sensore si attiva e il razzo spegne i quattro motori esterni. Cinque secondi di movimento per inerzia ed ecco che le cariche esplosive dell'anello di congiunzione esplodono allontanando lo stadio S-II . Trascorrono altri cinque secondi, necessari per evitare un contatto tra i due stadi, gli ugelli dello stadio S-IVB sprigionano il loro inferno di gas e quel che rimane del missile accelera dolcemente per 2 minuti e 30 secondi, fino a aggiungere quegli 820 m/s mancanti alla velocità obiettivo.

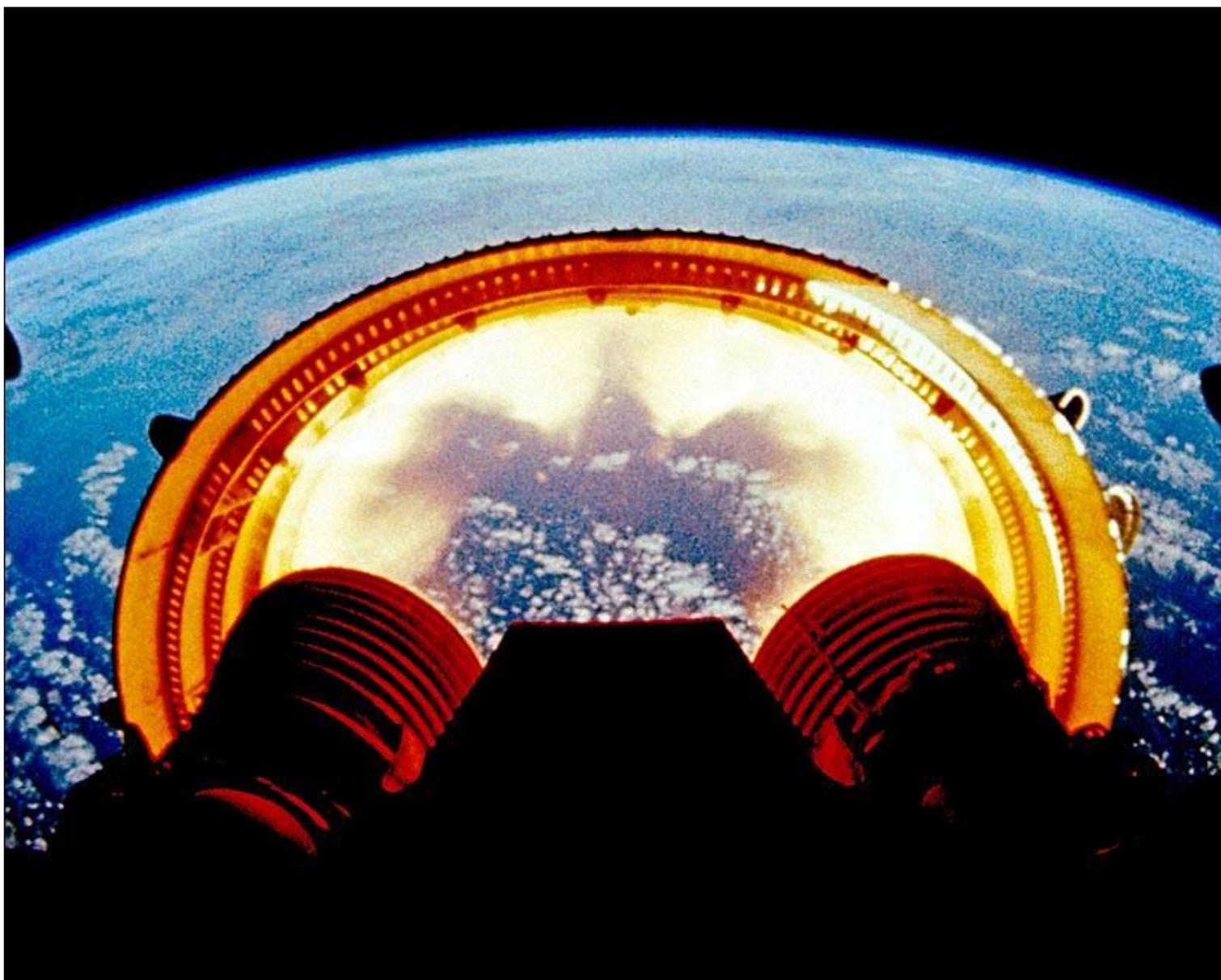
Sono trascorsi 11 minuti e 45 secondi dal decollo quando i motori del S-IVB cessano di spingere. Gli astronauti nel loro modulo di comando viaggiano a 7.800 m/s e sono entrati in orbita stabile. Così come previsto dall'equazione del razzo di Tsiolkovskij.

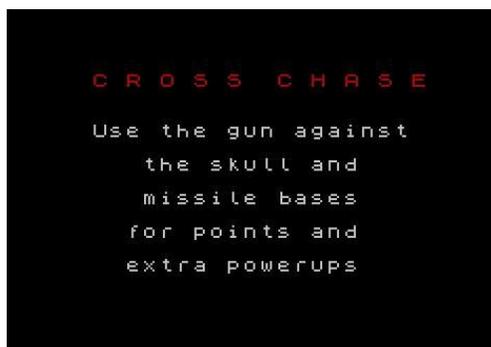
Il piccolo Konstantin non ha mai visto coronato il proprio sogno di vedere l'uomo abbandonare il pianeta ma ebbe modo di manifestare a più riprese la sua ferma convinzione che ciò sarebbe accaduto, perché inevitabile. Il Governo russo gli diede ragione, istituendo nel 1933 "l'Istituto per la ricerca scientifica del motore a reazione" del quale definì un programma sistematico di ricerca.

Come altri geni della nostra epoca, Konstantin Tsiolkovskij non è stato solo un talentuoso scienziato portato naturalmente per la sistematizzazione e comprensione dei fenomeni. Fu anche e soprattutto un visionario che inseguì un'idea di umanità perfettibile e quindi naturalmente protesa al proprio perfezionamento, destinata a conquistare mondi esterni e a sviluppare continuamente le proprie conoscenze. Non si imitò a sognarlo ma si adoperò per consegnare ai posteri gli strumenti per raggiungere lo scopo e un progetto di società. Con il suo modo pragmatico e sistematico di procedere, elaborò un piano che avrebbe portato l'uomo a conquistare le stelle passando per la costruzione di stazioni spaziali, e razzi spinti dall'energia nucleare. Il suo fervore arrivò anche a prefigurare la visione utopica di una civiltà eugenetica, costruita per impedire agli esseri umani meno dotati di procreare in favore di una selezione artificiale dei migliori. Idee che, come sappiamo, ebbero tutt'altra portata e

conseguenza nella Germania della seconda guerra mondiale. Eppure mai, in nessun suo scritto o discorso pubblico e a differenza dei mille anni di storia dei razzi che lo hanno preceduto, quest'uomo ha citato o teorizzato o proposto un uso bellico della scienza che ha contribuito a fondare. Anzi, conscio che una visione così articolata avesse bisogno di essere trasmessa, compresa e accettata da tutti, Tsiolkovskij fu anche un abile narratore che ha lasciato brillanti e stimolanti racconti di fantascienza e che ha avuto un ruolo diretto nella sceneggiatura e nelle riprese di "Il viaggio cosmico", il primo film di fantascienza sovietico. I suoi scritti e la sua capacità di attrarre il lettore nelle infinite possibilità del volo spaziale, gli valsero una fama mondiale e le sue ricerche ispirarono la generazione di ingegneri come Korolëv, Goddard e Von Braun che a partire dagli anni '40 ci ha portato a uscire dai confini del sistema solare. Ma di questo finalmente parleremo la prossima volta.

Per aspera ad astra.

Fig. 7: Separazione finale. Lo stadio S-IV B si appresta a portare gli astronauti in orbita attorno alla Terra, prima di tentare l'impresa di dirigersi verso la Luna.

Progetti: Cross Chase

Un gioco estremamente multi-sistema di **Fabrizio Caruso**

Schermata del titolo!

Come rivista dedicata al Retrocomputing sentiamo il dovere di promuovere qualsiasi iniziativa o progetto legato a questo mondo. Dopo la rassegna dell'evento Firenze Vintage Bit 2017 nel numero 4, questa volta vogliamo portare all'attenzione del grande pubblico un'interessante iniziativa intrapresa qualche mese fa da Fabrizio Caruso. Vediamo di scoprire dalla viva voce dell'autore di cosa si tratta!

 # L'autore

Mi chiamo Fabrizio Caruso. Sono uno sviluppatore software e collezionista entusiasta di retro-computer. Ho circa 80 retrocomputer di svariate marche dai più comuni Commodore 64, Vic 20, Commodore 16, Sinclair ZX Spectrum ad alcuni modelli meno comuni come il Mattel Aquarius, il Commodore 116, Matra-Hachette Alice, etc.

Sono particolarmente interessato all'era 8-bit, cioè il periodo che va dalla fine degli anni 70 e l'inizio degli anni 90. Mi piace usarli e restaurarli. Ultimamente ho scoperto il piacere di programmare su queste macchine (su tutte!).

 # Il progetto

CROSS CHASE è il mio progetto personale il cui scopo è creare un gioco con grafica e sonoro semplici ma che sia soprattutto

Versione CPC

divertente da giocare e, cosa ancora più importante, che possa girare su "tutti" i sistemi 8 bit storici con abbastanza memoria (5k): computer, console, ibridi console-computer, calcolatrici scientifiche e handheld e per i quali esista un cross compilatore ANSI C.

Il mio progetto è open source e può essere seguito sulla pagina GitHub: https://github.com/Fabrizio-Caruso/CROSS-CHASE dove, oltre al codice sorgente, metto a disposizione anche i binari già pronti per più di 80 computer e console (nella sezione release).

 # Il framework

Il progetto ha richiesto la scrittura di un framework per la scrittura di giochi 8 bit "universali" che consente la scrittura contemporanea di un gioco 8 bit su un centinaio di piattaforme 8 bit diverse. Al momento attuale il framework è ad uno stato iniziale ma ha già la proprietà più importante: il suo codice è indipendente dal gioco che sto sviluppando.

Quando il framework sarà maturo, sarà possibile usarlo per creare rapidamente altri giochi 8 bit universali SENZA dover fare porting o riscrittura del codice per ogni macchina.

 # Cosa mi ha fatto cominciare

L'idea nacque 9 mesi fa con la mia scoperta del toolkit di sviluppo CC65 per sistemi basati su CPU 6502 e suoi derivati (come il Commodore 64, Commodore 16, Commodore Vic 20, Atari 400/800, NES, etc.). Questo toolkit mette a disposizione un linguaggio comune (ANSI C) e alcune API per diversi computer e console.

Cominciai modificando il classico esempio "hello world" che viene fornito con toolkit e dopo circa 1500 commits (modifiche) ho ottenuto con un gioco arcade completo che gira su un centinaio di sistemi totalmente diversi e incompatibili tra loro.

 # Tools di sviluppo

Chiaramente CC65 non sarebbe bastato per coprire tutti i sistemi 8 bit. Ho quindi dovuto usare altri toolkit simili per altri sistemi.

I principali tools di sviluppo che ho usato sono:

Versione Spectrum

- **CC65** - https://github.com/cc65/cc65
- **Z88DK** - https://github.com/z88dk/z88dk
per i sistemi basati su CPU Z80 (come il Sinclair Spectrum, MSX, Amstrad CPC, etc.)
- **CMOC** - https://perso.b2b2c.ca/~sarrazip/dev/cmoc.html per i sistemi basati su CPU 6809 (come Dragon 32/64 e CoCo 1/2/ 3) e la sua variante WinCMOC.

In totale attualmente uso 5 cross compilatori

- **cl65** (CC65)
- **sccz80** (Z88DK)
- **zsdcc** (variante di sdcc in Z88DK)
- **CMOC**
- **WinCMOC** (variante moddata di CMOC)

Sto sperimentando con altri cross compilatori come GCC for TI (per TI99/4A), GCC6809 e sdcc (sia "liscio" sia la versione del toolkit CPCTelera per Amstrad CPC).

Lo scopo è di supportare il più grande numero di sistemi ma con una regola ferrea: nessun porting o riscrittura del codice del gioco. Lo stesso codice deve essere usato per tutti i computer e console. Il codice specifico è

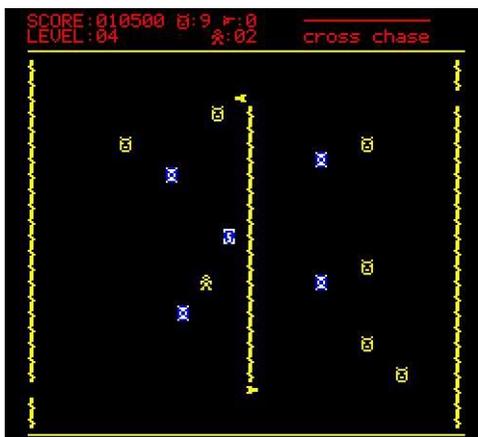
Versione MSX

invece nel framework e solo per alcuni dettagli dell'input/output.

 # 100 sistemi contemporaneamente supportati con lo stesso codice?

Questo è possibile perché:

1. Uso ANSI C che è un linguaggio universale che è ricompilato da vari cross-compilatori per produrre lo stesso gioco su sistemi diversi.

2. Ho creato dei livelli di astrazione nel mio

Versione Oric

codice (il framework!) così da poter inserire, in alcuni casi, del codice specifico per vari hardware senza modificare il codice del gioco.

3. Uso meta-codice per selezionare le porzioni di codice che servono per ogni sistema.

 # Il Gioco

L'idea del gioco è totalmente originale: il giocare è inseguito da dei nemici che può fare fuori facendoli schiantare contro delle mine che vede solo il giocatore o sparando contro loro se ottiene la pistola. Esistono 10 diversi power-up che danno poteri speciali. Alcuni dei quali sono nascosti e vanno scoperti compiendo specifiche operazioni (esempio: colpire o uccidere il teschio, etc.).

Non mi sono ispirato a nulla ma ho scoperto che esistono giochi simili. Il più simile è Robots: https://wiki.gnome.org/Apps/Robots che è incluso in Gnome. La differenza maggiore sta nel fatto che Robots è turn-based mentre il mio gioco è un gioco d'azione. Altra differenza importante è l'uso della pistola che serve a distruggere alcuni nemici e ostacoli.

 # Tre versioni

Lo stesso codice del gioco produce 3 diverse varianti del gioco che sono più o meno complete in maniera tale da poter girare su hardware anche molto limitati (anche soli 5k di ram).

Quindi il gioco esiste in versione:

- Versione **TINY**: nemici semplici e mine
- Versione **LIGHT**: aggiunta di alcuni power-up ed il teschio (nemico da uccidere con la pistola)
- Versione **FULL**: versione completa

 # Sistemi supportati

Il gioco supporta in principio tutti i sistemi a 8 bit comprese svariate configurazioni di memoria e video. Per il momento posso confermare il funzionamento su circa 80 sistemi. Aggiungo in continuazione nuovi sistemi, potete vedere una lista parziale nella tabella a fianco, ma vi rimando al sito GitHub sopra menzionato per informazioni più complete. Molti altri sistemi verranno supportati prossimamente!

 # Ringraziamenti

Pur essendo un progetto personale, sono stato aiutato da diverse persone. In particolare da Stefano Bodrato che è uno dei principali sviluppatori di Z88DK.

Sono anche stato aiutato da alcuni membri di varie "scene" 8-bit (in particolare da Simon Jonassen che sviluppa per CoCo e Dragon). Ringrazio anche Christian Groessler (sviluppatore CC65) per avermi dato alcuni consigli su Atari.

Versione Atari 800

A noi il progetto di Fabrizio e' sembrato subito molto interessante ed abbiamo colto al volo l'opportunita' di ospitarlo nella nostre pagine. Invito quindi Fabrizio a tornare ospite della nostra rivista nei prossimi mesi per fornire aggiornamenti e dettagli tecnici per meglio comprendere la complessita' del suo lavoro!

A cura di Francesco Fiorentini.

Versione FULL
- Luxor ABC80 32k
- Jupiter Ace 16k
- Apple //c
- Apple][e
- Mattel Aquarius with 16k expansion
- Atari 5200 (console)
- Atari 400/800 (color low resolution)
- Atari 400/800 (high resolution)
- Tangerine Atmos and Oric 1 48K
- Commodore 128 (native 40 column mode)
- Commodore 128 (native 80 column mode)
- Commodore 16/116/+4 (32k min)
- Commodore 64
- Commodore CBM 510
- Commodore CBM 610
- CoCo 1/2/3 and Dragon 32/64 (multiple versions)
- Amstrad CPC
- Galaksija 22k
- Gamate (console)
- Lambda 8300 16k
- CCE MC-1000 48k
- MicroBee
- MSX 32K (cassette and rom version)
- MTX
- Nascom computer series 32k
- NES (console)
- Ohio Scientific 1P 32k
- Philips P2000
- PC-6001 32K
- Commodore PET 16k
- Sam Coupe
- Sharp MZ series
- Sinclair Spectrum 48K
- Spectravideo SVI 328
- VG-5000 with 16k expansion
- Vic 20 with 16k expansion
- VZ 200 family with 32K
- Robotron Z 9001 32k
- Robotron Z 1013 32k
- Sinclair ZX80 with 16k expansion
- Sinclair ZX81 with 16k expansion

Versione LIGHT
- Luxor ABC80 16k
- Atari 400/800 (high resolution)
- Commodore 16/116 (unexpanded)
- CCE MC-1000 16k (unexpanded)
- MSX 16k
- Nascom computer series 16k
- Oric 1 16k (unexpanded)
- Philips P2000 16k (unexpanded)
- Spectravideo 318 16k (unexpanded)
- VG-5000 16k (unexpanded)
- Vic 20 with 8k expansion
- VZ 200 family with 16K
- Robotron Z 9001 16k
- Sega SC 3000 with 16k
- Atari Lynx 16k

Versione TINY
- Creativision (comp./cons. hybrid) 8k rom version
- Ohio Scientific 1P 8k
- Commodore PET 8k
- Sinclair Spectrum 16k (unexpanded)
- Commodore Vic 20 with 3k expansion
- Commodore Vic 20 (unexpanded)
- PCEngine (8k rom version)
- Mattel Aquarius with 4k expansion
- Epson PX-4/HC-40/HX-40
- Sharp X1

Chiusura ed anticipazioni…

di Francesco Fiorentini

Come scritto nell'editoriale, RetroMagazine cresce, non solo con la pagina Facebook ed il sito web, ma anche nei contenuti e nel numero di pagine.

Il numero 5 infatti si compone di ben 46 pagine! 46 Pagine ricche di informazioni e di nuove rubriche che speriamo siano di Vostro gradimento.

Abbiamo dato il via ad una nuova rubrica, RetroSpace, che unendosi alle altre gia' esistenti, RetroMath e Console 8bit ci accompagnera' per diverso tempo in questo nostro viaggio.

Abbiamo voluto dare spazio ad uno dei progetti che riguardano il mondo del Retrocomputing e ci siamo posti l'obiettivo di scovarne altri e di proporveli in modo da dare visibilita' ed ottenerne a nostra volta… ☺

A questo punto pero' e' inutile che mi dilunghi su questo numero, che gia' avete tra le mani, quanto invece e' giusto che vi anticipi cosa ci e Vi riservi il futuro.

La serie di articoli sulla programmazione dell'Atari 2600 a cura di **Giorgio Balestrieri** non si e' ancora esaurita e ci sono ancora delle cose che Giorgio ci vuole raccontare, ma soprattutto ci ha promesso di scrivere un gioco vero e proprio. Curiosi?

Torneremo di nuovo a parlare di Assembly del C64 insieme a **Marco Pistorio** e per non fare torto a nessuno, **Walter Pugi** ci introdurra' alle basi della programmazione dello ZX Spectrum.

Personalmente non ho mai posseduto uno Spectrum e ne sto cercando uno da diverso tempo, ma fortunatamente con gli emulatori si riesce a rimediare a queste lacune. Quindi oltre a rimediare alla lacuna Hardware spero di rimediare anche alla lacuna Software imparando a programmare lo Speccy.

Dante Profeta invece ci guidera' alla scoperta dell'architettura dell'Amiga e dei suoi modi grafici.

Alzi la mano invece chi ha mai sentito parlare del MK14, l'antenato dei Sinclair ZX? Devo ammettere la mia ignoranza, per fortuna **Alberto Apostolo**, collaboratore esterno di RetroMagazine ci racconterà' i dettagli di questo computer semi-sconosciuto.

E te? Non fai niente a questo giro? Non sia mai che resti con le mani in mano… Ve l'ho detto che la nostra mission e' quella di dare visibilita' ai progetti che scoviamo in rete o che ci vengono proposti. Bene, qualcuno sta usando un vecchio computer per le previsioni del tempo ed a me e' sembrata veramente una cosa interessante, quindi… Aspettate e leggerete!

Per il momento vi ho detto anche troppo…

Al prossimo numero di **RetroMagazine**!

Disclaimer

RetroMagazine
Anno 2 - Numero 5

Direttore Responsabile
Fiorentini Francesco

Immagine di copertina
Flavio Soldani

Marzo 2018